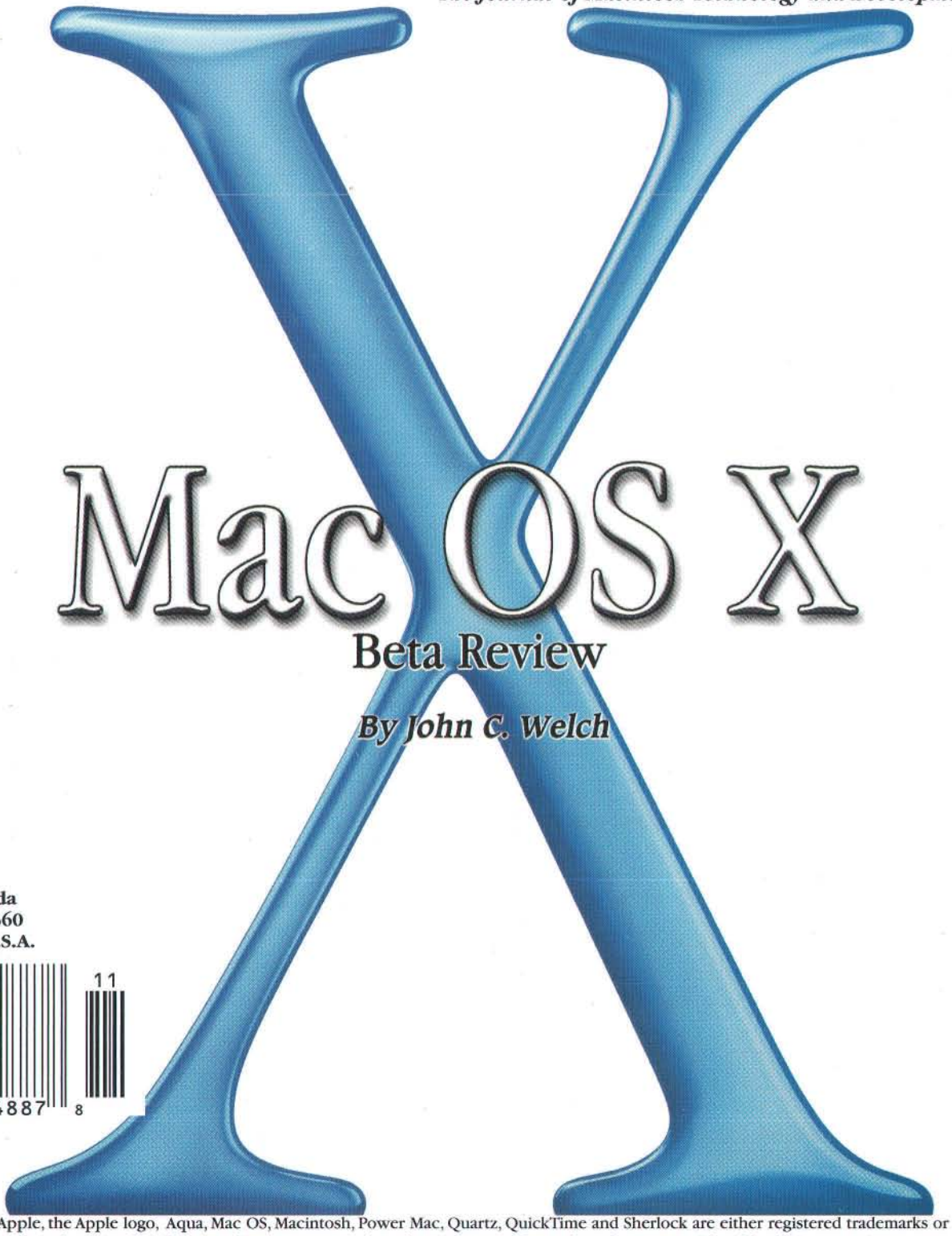
MacTech Magazine
Vol. 16, No. 11 • November 2000

MacTech®

The Journal of Macintosh Technology and Development

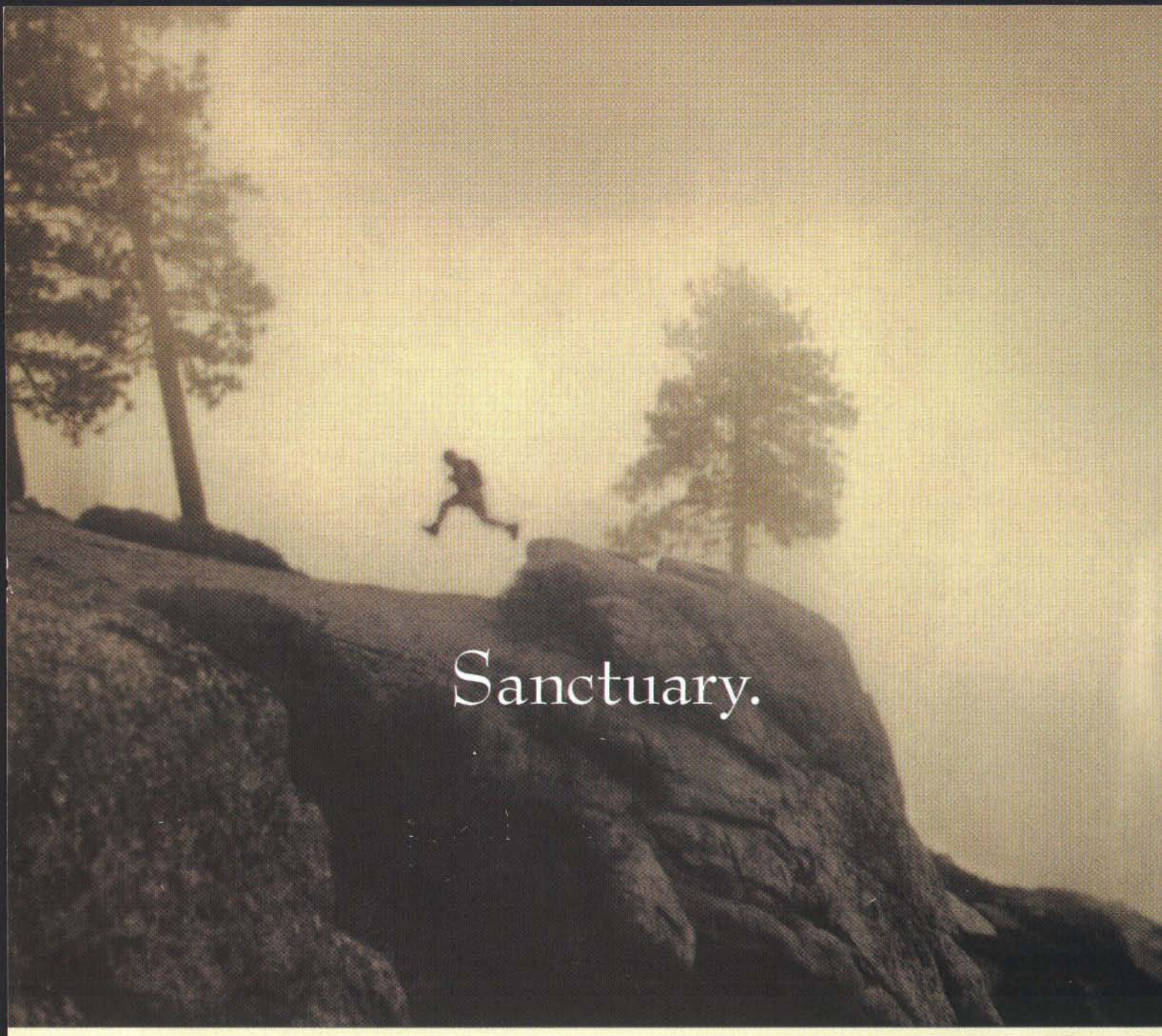


Mac OS X

Beta Review

By John C. Welch

$8.95 US
$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.



Move to digital.forest. The leading Internet Service Provider for the Macintosh community.

You need maximum flexibility, security and performance from your ISP. You'll find it with digital.forest. We're the proven provider of Macintosh Internet service.

Founded in 1994, our specialty is Mac systems and technologies. In fact, digital.forest is the world's largest Macintosh Hosting Provider. And our hard-earned knowledge of Mac server software is the broadest in the business.

We pioneered Macintosh Server Colocation and FileMaker Pro database hosting services. And our commitment to service keeps growing. So whether you choose us to host your files or house your servers, you've chosen the best.

Go with the industry leader. Discover the sanctuary of digital.forest.

digital.forest

877-720-0483

info@forest.net

www.forest.net

Call toll-free in the U.S. and Canada.
For international inquiries, please call 425-483-0483.

FileMaker Solutions Alliance Associate

FileMaker Pro is a registered trademark of FileMaker, Inc. The digital.forest logo is trademark or registered trademark of digital.forest. Inc.

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."
– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."
– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"
– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."
– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."
– Joe Zobkiw, author of ***A Fragment of Your Imagination***

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater,
1.5MB RAM, CD-ROM

Standard price: $256 (decimal)
Website price: $128 - $256
(Educational, quantity, or
other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red,
International Shipping

MATHEMÆSTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New in 2.0:

- **Very fast, HFS browser for viewing file tree of all volumes**
- **Extensibility for new Resorcerer Apprentices (CFM plug-ins)**
- **New AppleScript Dictionary ('aete') Apprentice Editor**
- **MacOS 8 Appearance Manager-savvy Control Editor**
- **PowerPlant text traits and menu command support**
- **Complete AIFF sound file disassembly template**
- **Big-, little-, and even mixed-endian template parsing**
- **Auto-backup during file saves; folder attribute editing**
- **Ships with PowerPC native, fat, and 68K versions**

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

MacTech®
The Journal of Macintosh Technology & Development

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 877-MACTECH

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@mactech.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

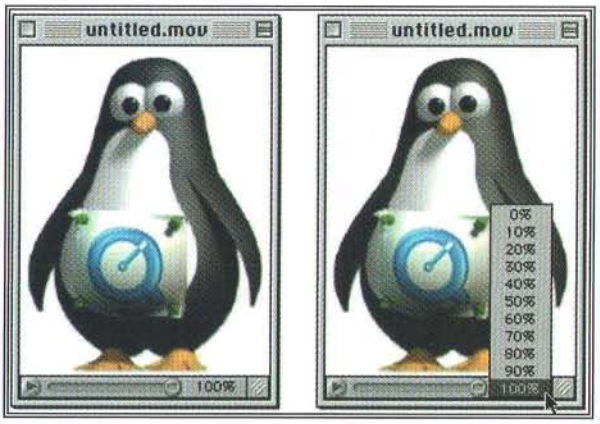
MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

November 2000 • Volume 16, Issue 11

Feature Articles



QuickTime Toolkit .....Page 60

Cover Story

Mac OS X Public Beta.....Page 34

Columns

By William Porter, POLYTROPE, Houston, Texas

4D/WebSTAR Summit 2000

An exclusive MacTech report

The 4D/WebSTAR Summit was held October 4–8 in San Diego, California. The weather in San Diego was uncharacteristically gloomy throughout the Summit, but inside the U.S. Grant Hotel, the atmosphere was sunny and warm.

This was the largest gathering of 4D developers ever. According to Mike Erickson of Automated Solutions Group, co-sponsor with 4D, Inc. of this year's Summit, there were almost 450 developers in attendance, from two dozen countries. A dozen third-party vendors displayed their products or services during the Summit. The three-day program consisted of over 70 sessions. Some of these were presented by personnel from 4D, Inc. or WebSTAR and a few were given by third-party vendors who discussed their products. But most were presented by the several dozen independent developers who ensured that the program was grounded not in marketing but in practical problem solving. Best of all, there were more new users than ever before. There was a special track for 4D beginners and a *free* one-day seminar taught by Liz Delgado designed to bring newbies rapidly up-to-speed.

4D, Inc., president and CEO Brendan Coveney, in his keynote address, reported that the state of the company was good indeed. NASA recently purchased an agency-wide site license for 4D. Projected revenues for fiscal year 2000 come to $28 million, with operating costs of only $6 million. Revenues have grown 80% in the last two years. Increased expenditures on marketing are paying off: 4D, the oldest RDBMS for the Mac, is once again a visible and respected presence in the world of Mac OS database development.

WebSTAR

The acquisition of Starnine, makers of WebSTAR, in March 2000, overnight made 4D, Inc. a major web player. If you weren't paying attention at the time, WebSTAR is the software that the U.S. Army moved its web site to a year ago, after it dumped Microsoft IIS because of its weak security. Exactly how WebSTAR will play into 4D's long-term product strategy remains to be seen. While 4D already has outstanding proprietary tools for web-serving databases, it seems reasonable to expect closer integration between 4D, Inc.'s database and web-server products in the future. But C.J. Holmes, formerly of Starnine and now 4D, Inc.'s director of engineering for WebSTAR, assured me that WebSTAR is not going to be absorbed into 4D as a component, or vice versa. And in an exclusive interview with MacTECH, Brendan Coveney confirmed this, saying that 4D, Inc. remains an "open-systems company" and that WebSTAR will continue to work well with all databases on the Mac. This will be good news for FileMaker Pro developers using Lasso or the FileMaker Web Companion and for users of other back ends like Valentina or PrimeBase.

No release date was given, but attendees were given some hints about what to look for in WebSTAR 5. It was described as a well-behaved application with an instinct for self-preservation; if a child process dies for any reason, the parent process will start a new process automatically. WebSTAR 5 will ship with a bunch of new features, including support for Perl; support for multiple processors; more plug-ins and services such as calendars, forms processing, etc. Tests run with WebBench show spectacular improvements in speed: WebSTAR 4.x runs around 50 connections per second. WebSTAR 5 has been tested at over 420 connections a second or 37 million connections a day. These are some serious web-serving numbers. Finally, they have WebSTAR 5's core features running under OS X already as a BSD application.

In honor of the company's new product, there was a special track devoted to WebSTAR and other issues of interest to web administrators; about 50 attendees registered for that track specifically. One web

Continued on page 96

The key to thinking different...

Works on the iMac!

The New MacHASP USB Key!

Question: Is MacHASP USB* a software security key or a sales tool?

Answer: It's both!

MacHASP USB is the world's first software protection key for the iMac. It gives you sophisticated license enforcement and comprehensive protection against illegal use ... in other words, real security.

Then it gives you a big selling advantage.

With MacHASP USB, you can license multiple software modules and applications. You can instantly unlock or upgrade them in the field. Plus you can freely distribute demos.

Bottom line: MacHASP USB locks out illegal users and unlocks your full sales potential – without getting in anyone's way. Call now to request a Developer's Kit and our newly published guide to USB features and benefits.

*For all USB-equipped Macs running an OS with USB support. Fully compatible with the ADB version of MacHASP.

Mac OS

Accelerated for Power Macintosh

USA: 1-800-223-4277
212-564-5678
Int'l: 972-3-6362222

www.aks.com/mt

ALADDIN®
KNOWLEDGE SYSTEMS LTD

Mac software protection provider, Micro Macro Technologies, becomes part of Aladdin, giving its customers even better service from the number one name.

By John C. Welch

Networks 201 pt. 5

Layer 3: The Network Layer

Refresh

From the first article in our series, we recall that Layer 3, the Network Layer, is responsible for handling network connections that exist past the next object in line. In other words, Layer 3 is the routing layer. This is the layer that handles packet transmission over subnets, and between different types of networks. Layer 3 is not required in all circumstances. If you are using a network that does not route, or does not need routing information, Layer 3 may be very thin, or nonexistent. This is also the lowest level of the OSI model that communicates in an end-to-end fashion. This means that as far as Layer 3 is concerned, there are no Layers 1 or 2, only other machines running Layer 3 protocols. This is what we will talk about this month, so into the Fray!

Layer 3

As we noted above, the Network Layer deals with a bigger scope than the Data Link Layer. Where the Data Link Layer is concerned with getting frames from wire end A to wire end B, the Network Layer is concerned with getting the packet from the source to the destination, regardless of how many wires, routers, or other points in between the source and the destination. Like all the other layers, the Network Layer provides services to the Layer above it, in this case, the Transport Layer. This interface between the two layers is often the boundary of the network subnet, or the boundary between the customer, (the Transport Layer and up), and the carrier, (Network Layer and down.) To do this, the Network Layer services were designed with three primary goals:

1) Network Layer services need to be independent of the subnet technology. That is, the services provided by the layer need to not care about whether the subnet is a TCP/IP, AppleTalk, or any other protocol.

2) It needs to shield the Transport Layer from the number, type, and topology of the subnets present. The Transport Layer does not need to know any of this, as this is what the Network Layer does. All the Transport Layer needs to do is hand off information and data to the Network Layer, and let the Network Layer do it's job. This is in keeping with the general idea of each Layer having a specific purpose within the OSI model.

3) The network addresses used by the Transport Layer should be part of a uniform numbering plan, regardless of the scope of the network. In other words, the transport layer shouldn't have to deal with how the network is addressed, or the scope of those addresses. Just that the addresses are there, and apply across the network.

To accomplish these goals, there are two points of view on how to do this, and both work well within their areas. The first point of view is that of the Internet community, and says that the only thing the subnet, and by extension, Layer 3 should be doing is pushing and getting bits. This takes the argument that the subnet is inherently unreliable, and that any error control and flow control need to be handled by the endpoints, or hosts. The Network Layer here, should be connectionless, and use only the smallest amount of network primitive commands, (SEND PACKET, RECEIVE PACKET, and

John Welch <jwelch@aer.com> is the Mac and PC Administrator for AER Inc., a weather and atmospheric science company in Cambridge, Mass. He has over fifteen years of experience at making computers work. His specialties are figuring out ways to make the Mac do what nobody thinks it can, and showing that the Mac is the superior administrative platform.

not much else.) The reason that the layer should do no flow or error control is because the hosts are going to do that anyway, and besides, who knows where the packets really go in between points A and B with any reliability. To support the multiple paths packets may be taking, each packet needs to carry the full addresses of the source and destination.

In the other corner is the point of view of the telecommunications industry. This says that the subnet should be reliable, and should be connection oriented. There should be some error and flow control in the subnet, and all data transfers should have certain basic properties along the following lines:

1) Before sending or receiving data, a connection is set up between the source and the destination. This connection creates a path between the two, and is a temporarily static path that encompasses any midpoint devices. This connection has a unique identifier that helps route packets.

2) Once the connection is set up, then the two ends negotiate parameters, quality, cost, etc.

3) All communications are bi-directional, and packets are delivered in sequence.

4) Flow control is provided automatically to keep from overloading one or both ends.

5) Once it is no longer needed, the connection is torn down, and all used buffers are flushed.

The real difference between the connectionless and connection - oriented arguments is where the complexity of the layer is handled. In a connectionless protocol, the end points deal with all the complexities of the network. This is because computing power is cheap, and it is easier to upgrade end nodes than major intermediary devices. Also, some functions, such as real-time oriented applications are far more concerned with speed of delivery, rather than accuracy of delivery. The connection - oriented folks argue that the subnet should help provide reliable, trouble-free service, and the end nodes shouldn't have to run complex Transport Layer protocols. In addition, there is a point to be made that real-time data does just as well in a reliable connection as in a connectionless service, and that it is easier to provide certain real-time information atop a reliable connection-oriented protocol.

In the end, both are used, depending on the application's needs. File transfers want a reliable connection, to avoid data corruption, whereas live video feeds prefer to drop a frame or two, while still keeping the stream running, without the overhead of resending multiple packets.

Connection – oriented services

These work primarily by creating virtual circuits that act as temporary paths between two end nodes. The idea here is to avoid having to create, or even look for a new route for every packet that is transferred. Instead, when a connection is established, a route between the two end nodes is created and stored, to be used for all traffic for the duration of that connection. Once the connection is taken down, the virtual circuit is also terminated. This has the effect of requiring a lot more out of the intermediary devices on the subnet. Routers must maintain an entry for every virtual circuit that is using it. They must check every packet for the virtual circuit number, so they can determine where the packet goes next.

When a new connection is created, the first unused virtual circuit, (VC) number is used. It is important to note that these numbers are of local significance, not global. This avoids having to synchronize every connection with every other connection to avoid VC number conflicts. Another issue with VC numbers is when a connection is initiated by both ends at once. This leads to two adjacent routers creating a duplex circuit that could have conflicting, (identical) VC numbers. At this point, the routers don't have any way to tell which way the packet is moving. One of the ways this is avoided is to use simplex connections.

The advantages to VCs are that the addressing is much simpler, relying on VC numbers more than full-blown addresses. The routing ends up being similar, because once the connection is established, that is the route that all packets will take for the duration of the connection. VCs also help with bandwidth needs, because part of the connection process is quality negotiation, so if need be, bandwidth can be reserved by the connection before the first packet is moved.

However, if the data needs of the connection are small, the overhead in setting up the VC can often be not worth the effort involved. Also, if one of the routers on the VC goes down, then the connection is broken, and has to be re-established. In fact, *all* the connections being serviced by that router are dropped, and have to be re-established.

Connectionless services

These are also known as datagram networks, as that is the name used for the packets in this type of network. Each datagram contains the complete addresses of its sender and recipient. There is no connection establishment, nor is there a route established for that data either. Indeed each datagram can go a different way than the datagrams in front of or behind it.

This has the advantage of being a more reliable method of data delivery if the subnet quality is unknown, or not reliable. Since each datagram is independently routed, no one device can destroy the entire delivery. The downside to this is that since every datagram is independently routed, the routing becomes much more complicated than for a VC. This also makes congestion and flow control difficult.

Routing

We said earlier that one of the primary functions of the Network Layer is that of routing, or getting packets from source to destination, regardless of network types and the number of nodes in between. The methods and algorithms involved in routing are numerous and complex, so we will deal with the simplest, so as to give you an idea of how they work, without going in to too much detail. (There are books written on routing algorithms, so if you would like to get into more detail, a visit to the computer section of a well-stocked bookstore can get you all the detail you would wish for, and then some.)

The routing algorithm is what decides how a packet will travel from a given router. If datagrams are used, this decision is made for every packet. If VCs are used, then this decision is only made during the connection establishment, and the packets follow this route. This type of VC routing is also called session routing, as the route is used for the entire session. No matter which type of routing is used, there are certain goals for any routing algorithm: correctness, simplicity, robustness, stability, fairness, and optimality.

The first two items are fairly obvious. The algorithm must be correct, otherwise, the packets will never be delivered correctly. It must also be as simple as possible, so that it can be fast enough to handle the loads placed upon it. The third property, robustness is not as obvious, but some routers are in place for years at a time. The algorithm used by a router must be able to handle failures by the other devices it directly deals with, changes in topology, protocol, numbering scheme, etc. It must be able to do this without requiring human intervention or attention as well. Stability is also somewhat obvious. The algorithm must not cause problems due to the way it functions, otherwise it is not useful.

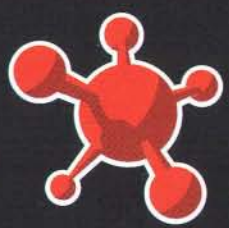
The final two are harder to reconcile with each other. Fairness dictates that no one part of the subnet be used to the point of saturation, yet choosing a route based solely on the optimal route may indeed cause this to happen. Even optimization can result in conflict, as minimizing packet delay does not always maximize network throughput. To help with this, and to deal with fairness, most algorithms concentrate on minimizing the number of hops a packet must make. This helps minimize delay while maximizing utilization.

While there are many algorithms, they all fall into two basic camps, static and adaptive algorithms. Static algorithms are decided outside of the router, and either downloaded to the router when it is booted, or manually entered on the router. If you have ever manually entered routes on products such as IPNetRouter, or SoftRouter, that is a type of static routing. Adaptive algorithms change routes based on information received from adjacent devices that inform them of the opening of a new route, or the closing of an existing one. These maintain their own routing tables, and do not require manual intervention to update themselves.

REAL PEOPLE, REAL SUPPORT.

MacTank provides technical support solutions for Macintosh application developers. We support your end users so you have time to concentrate on marketing and development. Bring your applications to OS X from Windows or NeXT and we will do the rest.

For pricing and information call 877-751-2300, option 2.



MACTANK
THE TECH SUPPORT ALTERNATIVE

Visit us at www.mactank.com

Algorithm Examples

Of the static algorithms, **flooding** is the probably the simplest. In a flood routing setup, an incoming packet is sent out on every single line the router has except for the one it came on. Now, obviously, the potential for bringing down a network through a potentially infinite number of packets on the network. So there are some techniques to avoid this, such as inserting a hop counter in the header of each packet, decrementing it each time it passes through a router, and discarding the packet once the hop count is equal to zero. Another technique is to set up each flooded packet with a sequence number. The source router then has to provide the subsequent routers with a packet list, so they know which packets have been flooded, and they are not re-flooded. Another variation is **selective flooding**, where packets are only flooded in the appropriate direction. (i.e., a westbound packet is not flooded back east.) Although flooding may seem to be of little use, for the military, or other organizations that need to be able to bypass dead, or blown up routers, flooding is a quick, simple method to do just that. As well, flooding always chooses the shortest path, because it chooses every path. Consequently, if the flooding overhead is ignored, flooding actually produces the smallest delay of any algorithm.

Anther static algorithm is **shortest path routing**. Simply put, with this algorithm, the subnet is displayed as a graph, with each point on the graph representing a router or end node, and each segment on the graph a communications line between points. The algorithm then determines the shortest path, and sends the packet on its way. There are a number of ways to determine exactly what is meant by 'shortest'. The most common is to find the path that has the least number of hops. However, this can break down, especially when a two-hop path is a hundred miles, and a four - hop path is fifty miles. To avoid this, shortest path routers actually use hop count, geographic distance, queuing and transmission delays, etc. to find the true shortest distance. Each factor is given a weight, and that weight is used to find the shortest path.

The disadvantage to static routing is of course, that it's static. It cannot take advantage of improved conditions, or handle worse conditions. It can only route the way it knows. So much of today's routers use dynamic algorithms, that can adapt to current conditions on the network, without human intervention. Since these are much more complex than static routing, we will only look at one of them, **distance vector routing**.

Distance vector routing algorithms function by having each router keep a table, or vector with the best known distance to each destination, along with the associated lines. The routers update the vector tables by exchanging information with their neighbors. This type of routing is one of the oldest, being not only the original ARPANET routing algorithm, but also used as the RIP algorithm, and by DECNet, IPX, AppleTalk, and Cisco routers.

The vector tables maintain certain parameters about each route. The entry for each route has the line to be used for that destination, and the estimate of the time to that destination. This time can be a measure of the hops to the destination, time delays, queue lengths, etc. The router is also assumed to know the distance to each neighbor. If the metric is hops, then there is only one hop. If queue length is used, then the router analyzes each queue. If delay is used, then the router measures this.

Although distance vector routing works well on paper, the real world implementations can have problems, particularly where updates are concerned. Although distance vector routing reacts well to improvements in the subnet, it can take much longer to react to bad news. Especially if time delays are used, and a node or router is down, (giving it a time delay of infinity), propagating that throughout the subnet can end up taking an extremely long time, hence the name for the problem, 'count to infinity'.

Conclusion

There are a lot of uses for the Network Layer, most of which I have avoided, as they tend to get into specific protocol types, or network types, and I wanted to stay away from any one protocol. But if there is any sort of routing going on, regardless of protocol or network type, it is most likely being done at the Network Layer level. I hope that you have an idea of the differences between connection - oriented, and connectionless services, and also a basic understanding of routing, and routing algorithms. Again, I avoided getting into the math of the algorithms, as that could easily take up an entire magazine, and is of more use to those folks writing router software. If, as a network manager, you understand what a router is trying to do, and why, you will find that troubleshooting, and designing networks will be noticeably easier, and the reasons why networks need to be set up in a given fashion will probably make a lot more sense to you. Our next article will deal with the Transport Layer, which is not only at the heart of the OSI model, but of most other protocols as well. As always, I encourage you to delve into these things on your own as well, using not just my bibliography sources, but any other books you may find on the subject.

Bibliography and References

- Tannenbaum, Andrew S. *Computer Networks.* Third Edition Prentice Hall, 1996

Quickly Build and Deploy Powerful Data-Driven Web Applications

Lasso Studio and Lasso Web Data Engine™ Lead The Way.

Building custom and shrink-wrapped database-driven Web applications requires a whole new way of doing things. Having pioneered the Web Data Engine™ over three years ago, Blue World and the Lasso Web Data Engine consistently lead the way providing a feature set Web developers describe as "incredible." Build online stores, discussion forums, resource management systems and other demanding database-driven Web applications with unrivaled performance, ease, security, extensibility, control and flexibility. Develop using multiple languages—including LDML, CDML, Server-Side JavaScript, Java and XML— and deploy across multiple platforms. Your Lasso code works identically regardless to which database you're connected. Lasso solutions for FileMaker® Pro databases easily scale to big iron ODBC-compliant databases like Oracle, Informix, Sybase and more with little or no change. What's more, the Lasso Java Application Programming Interface (LJAPI) provides developers an easy-to-use Java-based API for unprecedented extensibility.

Find out why hundreds of thousands of websites rely on award-winning Lasso technology for their business critical Web data. Download a 30-day evaluation copy at www.blueworld.com/download/ or order securely online today at the Blue World Store at store.blueworld.com.

Lasso Product Line – The leading Web tools for Macintosh and beyond.

bring business to the internet™

blueworld

By Maarten Gribnau and Tom Djajadiningrat

Cubby: Multiscreen Desktop VR Part II

How to create an Input Sprocket driver for a 3D input device

SUMMARY

In this second part of our 'Cubby: Multiscreen Desktop VR' trilogy, we will introduce you to the art of creating a driver to read an Origin Instruments Dynasight input device. With the Dynasight, the position of the head of the user is established so that Cubby can display the correct images on its screens. The driver is created with InputSprocket, which is part of Apple's Game Sprockets API.

INTRODUCTION

In our previous articles about Desktop VR (Djajadiningrat & Gribnau, 1998; Gribnau & Djajadiningrat, 1998), we used the Pointing Device Manager (PDM) of QuickDraw 3D to serve our input needs. We promoted the PDM because it was intended to support 3D input devices. The PDM and InputSprocket have in common that they both separate the device dependent code from the application. They relieve application programmers from dealing with devices directly and provide an abstraction layer for input programming. The difference between the two is that InputSprocket is actually supported by device manufacturers. There are no QuickDraw 3D drivers available to our knowledge. For InputSprocket on the other hand, lots of drivers are available. In addition, InputSprocket has configuration management built in. A standard interface is provided for connecting devices to applications. For these reasons InputSprocket provides a better solution for Cubby's input needs than the PDM.

In this episode we cover programming with InputSprocket to create a device driver. In the next episode the application side will be covered. Together, the two articles cover the whole of InputSprocket. The driver we describe is for the Dynasight device. If you do not own one or plan to buy one, you might still be interested because the techniques explained here can be used for other input devices as well. Documentation on writing InputSprocket drivers is scarce, so you might pick up some facts here. To compile our code, you will probably need to download the latest version of InputSprocket from the Apple web site (see References section), which is currently version 1.7.3.

If you are familiar with programming drivers for InputSprocket, you can safely skip the next section and proceed with the following section about creating an InputSprocket driver for the Dynasight. If InputSprocket drivers are new to you, you can read the next section and be introduced to the basic concepts. If you have worked with InputSprocket before but not with drivers, you can also read the next section to learn how drivers and applications communicate.

INTRODUCTION TO INPUTSPROCKET

The goal of InputSprocket is to make life easier for game programmers who deal with input devices. There are an immense number of devices on the market with different

Maarten is always jealous of his co-author. The ease with which Tom creates a funny 'about the author' piece starts to become almost traumatic. He is now on the verge of refusing to finish his contribution to the final episode. You might try to inspire him with an email at: M.W.Gribnau@io.tudelft.nl.

There are some striking similarities between **Tom** and his Wallstreet PoBo. You need to push them really hard to wake them from sleep, if you persist it still takes ages before they finally get into action, and sometimes it appears as if they've woken up but really all they do is zombie around in some undefined state in which it is impossible to get anything sensible out of them. If they're up and running you can reach them at J.P.Djajadiningrat@io.tudelft.nl.

features. Trying to have a game support all these devices optimally can lead to severe headaches. Most games support input devices through emulation of the keyboard and mouse. But emulation does not suffice when supporting complex devices that have directional pads, levers, wheels, etc. For one, the game cannot take advantage of the extra input controls of complex devices and secondly, configuring the devices and connecting them to game controls is device specific. That means that game programmers should provide configuration capabilities for each device that they need to support.

This is where InputSprocket comes in. It solves these problems through an input device architecture that allows games developers to create games that can use a wide variety of input devices. Input device developers can use InputSprocket to build device drivers that provide a description of input device controls that the game can use to automatically configure its control options. The device driver can also provide a user interface that allows the user to change default control options. Therefore, game programming is made easier because a lot of device dependent code is moved from the application to the device driver.

The communication between InputSprocket drivers and games is based on elements. The **element** is a building block used to describe the capabilities of a device. Each control of a device is described with an element, so every device can be described with a set of elements. For example, a one-button mouse can be described with an element for the x-axis, an element for the y-axis and a button element. More complex devices may require more elements but are handled in the same way. An element is described by the following three pieces of data.

A **human readable identifier** is a string that the game can display to identify the element for the user during configuration. Examples of such identifiers are "trigger", "roll" or "move forward".

The element **kind** is a four-character sequence that indicates the type of data the element produces. For example, button elements will typically produce two-state values while axis elements produce continuous data. InputSprocket currently provides five basic element kinds:

- Button elements produce two-state data.
- Directional pad elements are nine state elements with an idle position and eight states corresponding to the eight directions on a typical directional game pad.
- Axis elements produce continuous data, either with or without a meaningful center. A symmetrical axis has a meaningful center, like the axis of a joystick. Axis elements such as a gas pedal or a brake do not have a meaningful center.

- Delta elements are like axis elements but instead of an absolute position or orientation, they produce Delta data, which indicates the distance moved relative to the previous position (e.g. mouse axes).

- Movement elements produce movement data that is given both as x-y axis data and directional pad data, allowing the game to use whichever is suitable. Note that in general you should use axis data instead.

There is a sixth element kind, the virtual element, which we will encounter later.

The **label** of an element gives the suggested use for an element. For example, a button element may be intended as the start button or the firing button. During configuration, a game uses labels to find the elements it requires for play.

As was mentioned, elements are the way applications communicate with drivers. There are two interfaces to pass data from drivers to applications using elements: the low-level and the high-level interface. In principle, driver programmers can decide which interface to implement. It is tempting to implement the low-level interface only, since it is the easiest to implement. However, drivers should provide both the low and the high-level interface, so that game application programmers can decide which interface they want to use.

Low-level Interface

The low-level interface is the simplest to implement when building drivers. Figure 1 shows a diagram of both the low and the high-level interfaces. The lower parts of the Dynasight Driver and the InputSprocket Extension represent the low-level interface. Cubby uses only the high-level interface and therefore connects the cameras to the high-level axis elements (Head X, Head Y and Head Z) exclusively. When the driver is loaded, it creates elements for every control it has. The driver is responsible for reading the data from the device and, as soon as it is loaded, should update its elements every time the device reports new data. Applications can connect via InputSprocket directly to the elements of the device and update the game state accordingly.

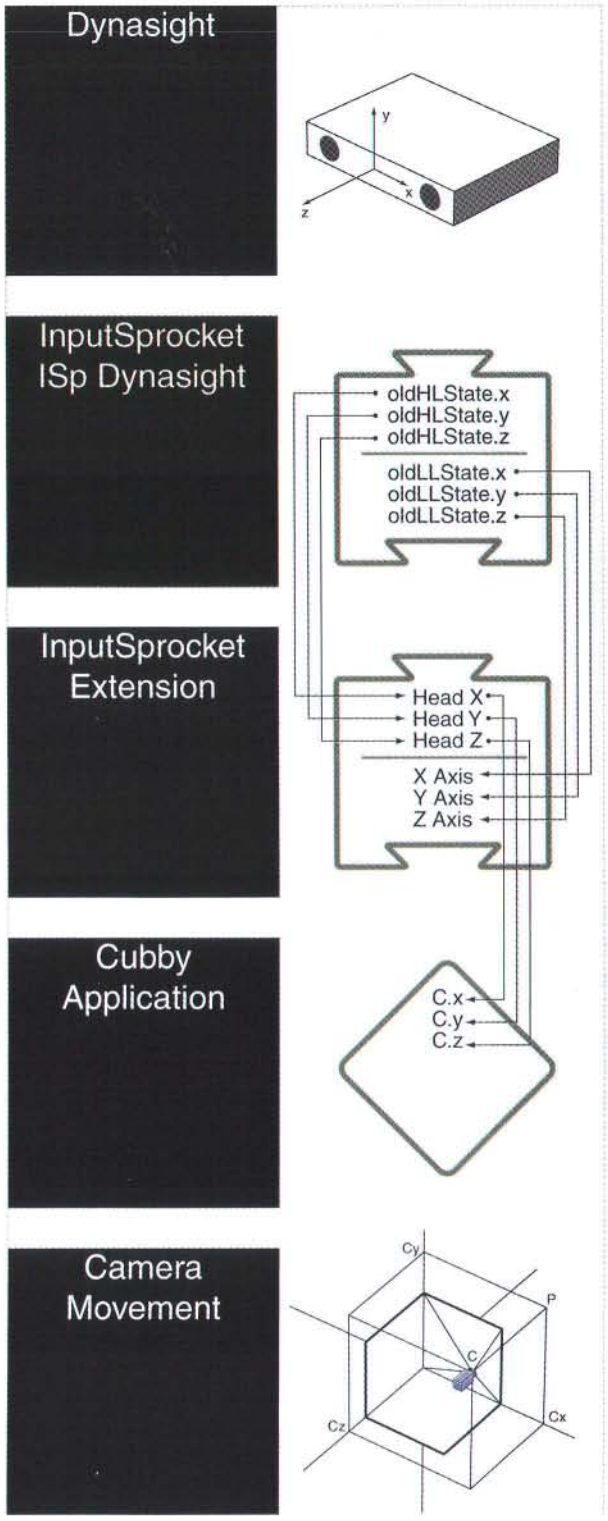


Figure 1. *Diagram of the low- and high-level interfaces.*

High-level Interface

In Figure 1, the upper parts of the Dynasight Driver and the InputSprocket Extension represent the high-level interface. The high-level interface is somewhat more complicated than the low-level interface but it has all the advantages that InputSprocket was meant to offer. Instead of connecting to the device's elements directly, games read data from the devices through **virtual elements**. These virtual elements are created by InputSprocket from the **needs** that the game has. Drivers are responsible for the mapping of needs to virtual elements. The high-level interface is only valid between the driver's Init and Stop callbacks that InputSprocket calls. Only the driver knows exactly how it has been configured, and when it is active, it is responsible for pushing data to those virtual elements for each need for which it is configured.

Configuration

The majority of driver callbacks are related to the Configure dialog. These function calls will only happen while the high-level is valid (i.e. ISpInit has been called without a subsequent ISpStop). When the application calls ISpConfigure, a dialog is presented with a scrolling list of devices with one device selected. Figure 3 shows this dialog with our Dynasight device selected. The popup menus show the current connection of the Dynasight x- y- and z-axis to a game's roll, pitch and yaw controls. The selected device is responsible for the primary pane of the dialog, handling all events and drawing related to that area. The GetSize function is called to determine the preferred and minimum sizes of the pane area used by the device. If the pane area will not fit on any available display device, then that device is removed from the high-level list. The BeginConfiguration function is then called for each device. The configure dialog is resized and shown. The GetIcon function is called for each device to determine the icon to be shown in the scrolling list. Then the Show function is called for the selected device. This is the time to call AppendDITL for the resources to be displayed in the primary pane. All events returned in the dialog filter are passed to the HandleEvent function to give the device a chance to handle them. If update events are not handled (the recommended option to avoid extra flicker), InputSprocket will make the device Draw function call from within a BeginUpdate/EndUpdate pair. Unhandled mouse clicks will be passed to the Click function. If the device called AppendDITL to add items to the dialog and those items are returned by InputSprocket's ModalDialog call, then it will call the DialogItemHit function.

The device should maintain a 'dirty' variable that is set whenever any configuration information is changed, and returned and cleared when the Dirty call is made. When a different device is selected in the list, the old device receives a Hide function call and the new device a Show function call. When the dialog is closed, every device receives an EndConfiguration call.

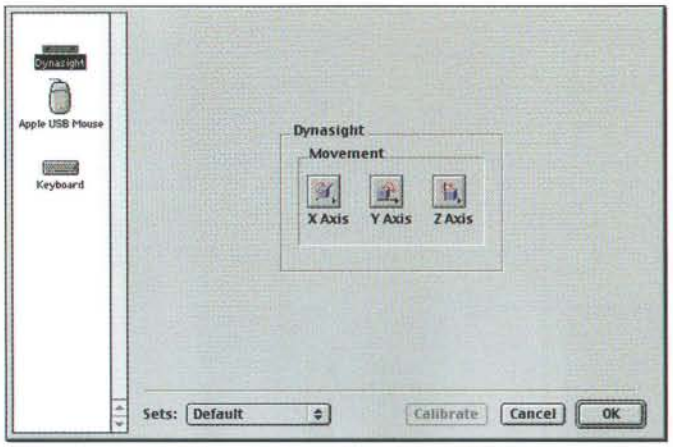


Figure 2. *InputSprockets's configuration dialog box with the Dynasight device selected.*

Procedure

Figure 3 illustrates the sequence of events when a game is using InputSprocket. The initiating calls are shown on the left and the driver's responses are found on the right. There are four major activities. Low-level initialization is entered when an application calls ISpStartUp. InputSprocket will then load the driver and call its major entry point. In response, the driver should look for devices present on the system and create InputSprocket devices for each active device. If a device was created, InputSprocket will call the driver's Meta handler to find out where the driver's entry points are. If no InputSprocket device was created, the driver is unloaded. High-level initialization is entered when the application calls ISpInit. InputSprocket passes the needs and virtual elements to the driver. In response, the driver stores them and does an auto-configuration. This means that the driver will try to find an optimal match between the needs of the application and the elements the device has. When the high-level interface is valid, the application can call ISpConfigure. This starts the configuration activities explained in the previous section.

There are several guidelines that should be followed during auto-configuration. If a previous device fulfilled a need and the kISpNeedFlag_NoMultiConfig bit is set in the need structure for that need, the device should not attempt to auto-configure to the need. The driver should indicate that it is fulfilling a need so that devices queried later know that the need is taken. More guidelines can be found in Apple's documentation of InputSprocket (see References section). After auto-configuring, the device should immediately push initial values to the corresponding virtual elements and from that point push data to those virtual elements whenever data is pushed to the elements to which they are configured.

When the application calls ISpStop, the validity of the high-level interface ends. The drivers should stop pushing data to the virtual elements and dispose the needs and the virtual elements. Calling ISpShutdown within the application causes InputSprocket to call the driver's termination routine. In response, the driver should stop pushing low-level data and dispose the InputSprocket device.

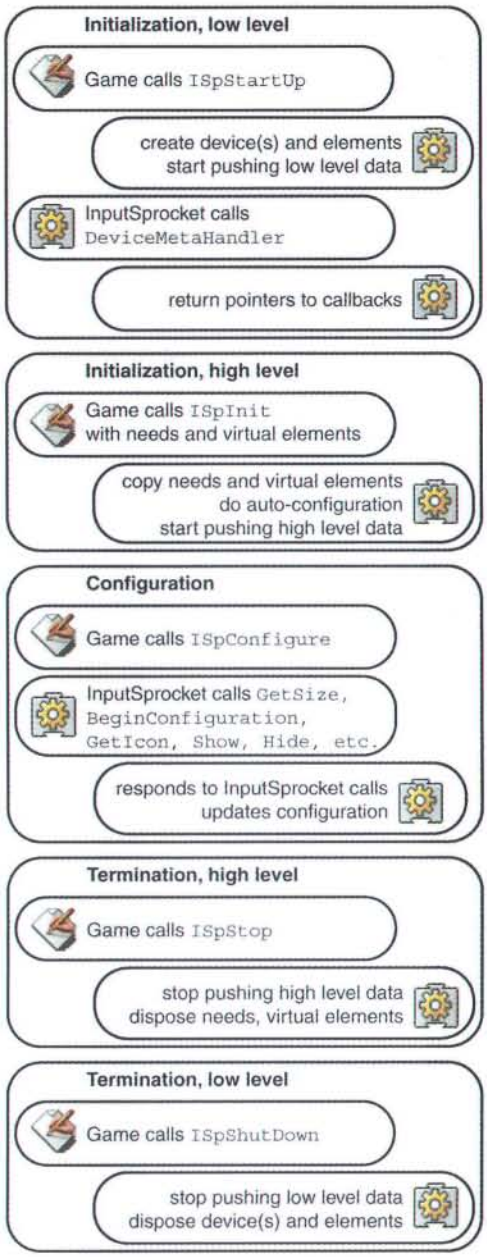


Figure 3. *The sequence of events in communication between driver and game.*

CREATING AN INPUTSPROCKET DRIVER FOR THE DYNASIGHT

Now that we have a basic understanding of how InputSprocket works, we will dive into the coding of a driver and illustrate how we can put this knowledge to use. We take the Dynasight device as example and start with the real driver functionality: the initialization and transfer of data. The configuration of the driver will be covered in the next section. The Dynasight device is an infrared tracking device that is used in Cubby to track the head position of users. The 3D position data is made available on the serial port. In the

following discussion, we will not go into the details of reading the position data from the serial port. If you are interested in that part of the driver, you can read our previous article about Desktop VR (Gribnau & Djajadiningrat, 1998) and/or you can have a look at the code accompanying this article.

Before we begin, we should make some general remarks about developing InputSprocket drivers. An InputSprocket driver is a shared library with a specific file type and creator ('shlb' and 'insp'). All drivers must be located in the same folder as the InputSprocket extension, which must be the Extensions folder. Being a shared library, it takes some extra effort to debug a driver. First of all, you should set the output directory of your development environment to the Extensions folder of the System folder on your startup disk, so that the compiled driver ends up there. Secondly, you should set a host application that is used to run the driver.

The Dynasight driver handles the six major tasks that all drivers should handle: shared library initialization, shared library termination, pushing data, 'high-level' Init, 'high-level' Stop, and the configure dialog user interface. The code for shared library initialization is shown in Listing 1. This routine is called when InputSprocket loads the driver. The important parts are that the driver checks whether a certain version of InputSprocket is available. If version 1.2 or greater is not available, the driver returns without creating any InputSprocket devices. In that case, InputSprocket unloads the driver. The other important point in this listing is that the driver retrieves the file specification of the shared library. The file specification is used in the driver initialization routine to locate the resource file.

Listing 1: ISpDriverDynasightMain.cp

__myintialize

```
OSErr __myinitialize(CFragInitBlockPtr ibp)
{
  OSErr  err = noErr;
  FSSpec fileSpec = {0,0,0};
  NumVersion version;
  UInt32 inputSprocketVersion;

#if __MWERKS__
  err = __initialize(ibp);
  if (err != noErr) {
    return err;
  }
#endif

  // Require InputSprocket 1.2.0
  version = ISpGetVersion();
  inputSprocketVersion = * (UInt32 *) &(version);
  if (inputSprocketVersion < 0x01200000) {
    return err;
  }

  // Grab the file spec
  if (ibp->fragLocator.where == kDataForkCFragLocator) {
    // The shared library should always be located in the data fork
    fileSpec = *ibp->fragLocator.u.onDisk.fileSpec;
  }

  // Create IS device for each Dynasight device found on the system
  // If we don't create an IS device, IS will unload us
  DynasightInit(fileSpec);
  return err;
}
```

In Listing 2, it is shown what happens inside the initialization routine. The routine looks at all available serial ports for a Dynasight. If one is found, the driver's resource file is opened and a structure in a global array (fDynas) is filled with all the information for the device on the current port. After setting some default values in the structure, the InputSprocket device is actually created (by calling CreateDevice) and the Dynasight is put into action. Finally, after all ports have been checked, the resource file of the driver is closed.

Listing 2: ISpDriverDynasight.cp

DynasightInit

```
OSErr DynasightInit(FSSpec fileSpec)
{
  OSErr err;
  short resourceRef = -1;                 // ref to our resource file
  Boolean resourceFileOpen = false; // true if our resource file is open
  SerialPort port;
  UInt16 portID, numPortIDs;
  Boolean  portWithDynasight;

  fNumDynas = 0;
  numPortIDs = port.GetNumSerialPorts();
  for (portID = 0; portID < numPortIDs; portID++) {
    {
      Dynasight dynasight;
      portWithDynasight =
        dynasight.GetDynasightFoundOnPort(portID);
    }
    if (portWithDynasight) {
      // Open the resource file and check for errors
      if (!resourceFileOpen) {
        resourceRef = FSpOpenResFile(&fileSpec, fsRdPerm);
        err = ResError();
        if (err != noErr) {
          return err;
        }
        resourceFileOpen = true;
      }
      fDynas[fNumDynas].dynasight = new Dynasight (portID);
      fDynas[fNumDynas].fileSpec = fileSpec;
      fDynas[fNumDynas].oldLLState.xAxis = kISpAxisMiddle;
      fDynas[fNumDynas].oldLLState.yAxis = kISpAxisMiddle;
      fDynas[fNumDynas].oldLLState.zAxis = kISpAxisMiddle;
      LowToHighLevelState(&fDynas[fNumDynas].oldLLState,
        &fDynas[fNumDynas].oldHLState);
      fDynas[fNumDynas].axisIndexToNeeds[kAxisIndex_XAxis]=
        kUnsetIndex;
      fDynas[fNumDynas].axisIndexToNeeds[kAxisIndex_YAxis]=
        kUnsetIndex;
      fDynas[fNumDynas].axisIndexToNeeds[kAxisIndex_ZAxis]=
        kUnsetIndex;
      fDynas[fNumDynas].virtualElementsValid = false;
      fDynas[fNumDynas].active = false;
      // Create the device and store its reference
      CreateDevice(&fDynas[fNumDynas]);
      fDynas[fNumDynas].active = true;
      fDynas[fNumDynas].dynasight->StartRunning(
        MyDynasightCompletionProc, &fDynas[fNumDynas]);
      fNumDynas++;
    }
  }
  // Close the resource file
  if (resourceFileOpen) {
    CloseResFile(resourceRef);
  }
  return err;
}
```

The CreateDevice routine actually creates the device and elements for the low-level interface. The driver provides three axis elements, corresponding to the x-, y- and z-axis of the Dynasight device. In our driver we chose to set up the device using a resource (of type 'isdv'), as shown in Listing 3. This resource contains all the information of an ISpDeviceDefinition structure that is passed to the InputSprocket. Listing 4 shows the creation of an axis element.

Again, we chose to set up the axis elements from a resource (of type 'isel') that contains the information in an ISpElementDefinitionStruct structure which is needed to set up an element. After retrieving the element resource, yet another resource is retrieved. This is the element configuration resource (of type 'isei') that is used to configure the element. This part is optional. If the resource is not available or invalid, the element is still created in the subsequent ISpElement_New call.

Listing 3: ISpDriverDynasight.cp

CreateDeviceFromResource

```
OSErr CreateDeviceFromResource(
   short resId,
   UInt32 refCon,
   ISpDeviceReference *device)
{
   Handle h;
   OSErr err;
   ISpDeviceDefinition def;

   // Read the device resource from the res file
   h = Get1Resource('isdv', resId);
   err = ResError();
   if (err != noErr) { return err; }
   if (h == nil) { err = -1; }
   if (err != noErr) { return err; }
   // Copy the handle to a structure and release the handle
   BlockMoveData(*h, &def, sizeof(def));
   ReleaseResource(h);
   // Create the InputSprocket device
   err = ISpDevice_New(&def, (ISpDriverFunctionPtr_MetaHandler)
DeviceMetaHandler, refCon, device);

   return err;
}
```

Listing 4: ISpDriverDynasight.cp

CreateElementFromResource

```
OSErr CreateElementFromResource(
   short elementResId,
   short configResID,
   ISpDeviceReference device,
   ISpElementReference *element)
{
   Handle h1;              // handle to resource based element definition structure
   Handle h2;              // handle to resource based element configuration info
   OSErr err;
   ISpElementDefinitionStruct def; // actual element definition structure

   // Read the element definition struct from a resource
   h1 = Get1Resource('isel', elementResId);
   err = ResError();
   if (err != noErr) { return err; }
   if (h1 == nil) { err = -1; }
   if (err != noErr) { return err; }

   // Copy the handle to a structure
   BlockMoveData(*h1, &def, sizeof(def));

   // Read the configuration information from a resource (this may fail)
   h2 = Get1Resource('isei', configResID);
   err = ResError();
   if (h2 && (err == noErr)) {
      HLock(h2);
      def.configInfo = *h2;
      def.configInfoLength = GetHandleSize(h2);
   }
   // Create the InputSprocket element
   def.device = device;
   err = ISpElement_New(&def, element);
   // Release resources
   HUnlock(h2);
   ReleaseResource(h1);
   ReleaseResource(h2);

   return err;
}
```

When creating the device in Listing 2, a pointer to DeviceMetaHandler was passed to the ISpDevice_New routine. This routine is called by InputSprocket to retrieve pointers to all the callbacks in our driver. It is called after the initialization of the driver. Listing 5 shows a part of the DeviceMetaHandler. InputSprocket calls the routine with a selector. The routine selects the appropriate callback and returns it. This example returns only the pointers to our Init and Stop callbacks. In our driver, the Meta handler returns pointers dealing with the configuration as well. These will be listed in the next section.

Listing 5: Example of a device Meta handler

DeviceMetaHandler

```
ISpDriverFunctionPtr_Generic DeviceMetaHandler(
  UInt32 refCon,
  ISpMetaHandlerSelector selector)
{
  ISpDriverFunctionPtr_Generic function = NULL;
  ISpDriverFunctionPtr_Init funcInit;
  ISpDriverFunctionPtr_Stop funcStop;

  switch(selector) {
    case kISpSelector_Init:
      funcInit = (ISpDriverFunctionPtr_Init) Init;
      function = (ISpDriverFunctionPtr_Generic) funcInit;
      break;

    case kISpSelector_Stop:
      funcStop = (ISpDriverFunctionPtr_Stop) Stop;
      function = (ISpDriverFunctionPtr_Generic) funcStop;
      break;
  }
  return function;
}
```

The beating hart of the driver is listed in Listing 6. This is the routine that is called whenever there is new data available from the Dynasight. We passed a pointer to this routine when we started the Dynasight in Listing 2. In the routine, it is first checked whether the low-level interface is active and whether the Dynasight is producing reliable data. Then, the position read from the Dynasight is scaled between 0 and 1 and subsequently scaled between InputSprocket axis values kISpAxisMinimum and kISpAxisMaximum. This is a requirement of InputSprocket. The convention is to scale coordinates of devices between those boundaries. For instance, with a joystick, kISpAxisMinimum should be sent when the joystick is fully rotated to one end and kISpAxisMaximum when it is fully rotated the other way.

The scaled coordinates are pushed to the low-level elements with the three ISpElement_PushSimpleData calls but only when a coordinate has changed. Then, the low-level data is converted to high-level data (in our case copied) and send to the virtual elements when the high-level interface is valid.

Listing 6: ISpDriverDynasight.cp

MyDynasightCompletionProc

```
void MyDynasightCompletionProc(
  DynasightCompletionProcData* data)
{
  OSStatus err;
  AbsoluteTime time;
  TDynasightRecPtr dyna = (TDynasightRecPtr)data->data;
  TISpLowLevelState llState;
  TISpHighLevelState hlState;
  float xf, yf, zf;

  if ((dyna->active) && (data->dynasightStatus ==
      dynasightStatus_Track)) {
    time = ISpUptime();

    // Scale Dynasight coordinates between 0 and 1
    NormalizeDynasightPosition(&data->position, &xf, &yf,
&zf);
    // Put point between kISpAxisMinimum and kISpAxisMaximum
    scale = kISpAxisMaximum - kISpAxisMinimum;
    xf = kISpAxisMinimum + xf * scale;
    yf = kISpAxisMinimum + yf * scale;
    zf = kISpAxisMinimum + zf * scale;
    llState.xAxis = (UInt32)xf;
    llState.yAxis = (UInt32)yf;
    llState.zAxis = (UInt32)zf;

    // Push low-level data for low-level interface
    if (llState.xAxis != dyna->oldLLState.xAxis) {
      err = ISpElement_PushSimpleData(
              dyna->deviceXAxis, llState.xAxis, &time);
    }
    if (llState.yAxis != dyna->oldLLState.yAxis) {
      err = ISpElement_PushSimpleData(
              dyna->deviceYAxis, llState.yAxis, &time);
    }
    if (llState.zAxis != dyna->oldLLState.zAxis) {
      err = ISpElement_PushSimpleData(
              dyna->deviceZAxis, llState.zAxis, &time);
    }

    // Push high-level data for high-level interface
    LowToHighLevelState(&llState, &hlState);
    if (dyna->virtualElementsValid) {
      SetVirtualsData(dyna, &hlState);
    }

    dyna->oldLLState = llState;
    dyna->oldHLState = hlState;
  }
}
```

InputSprocket Driver Configuration

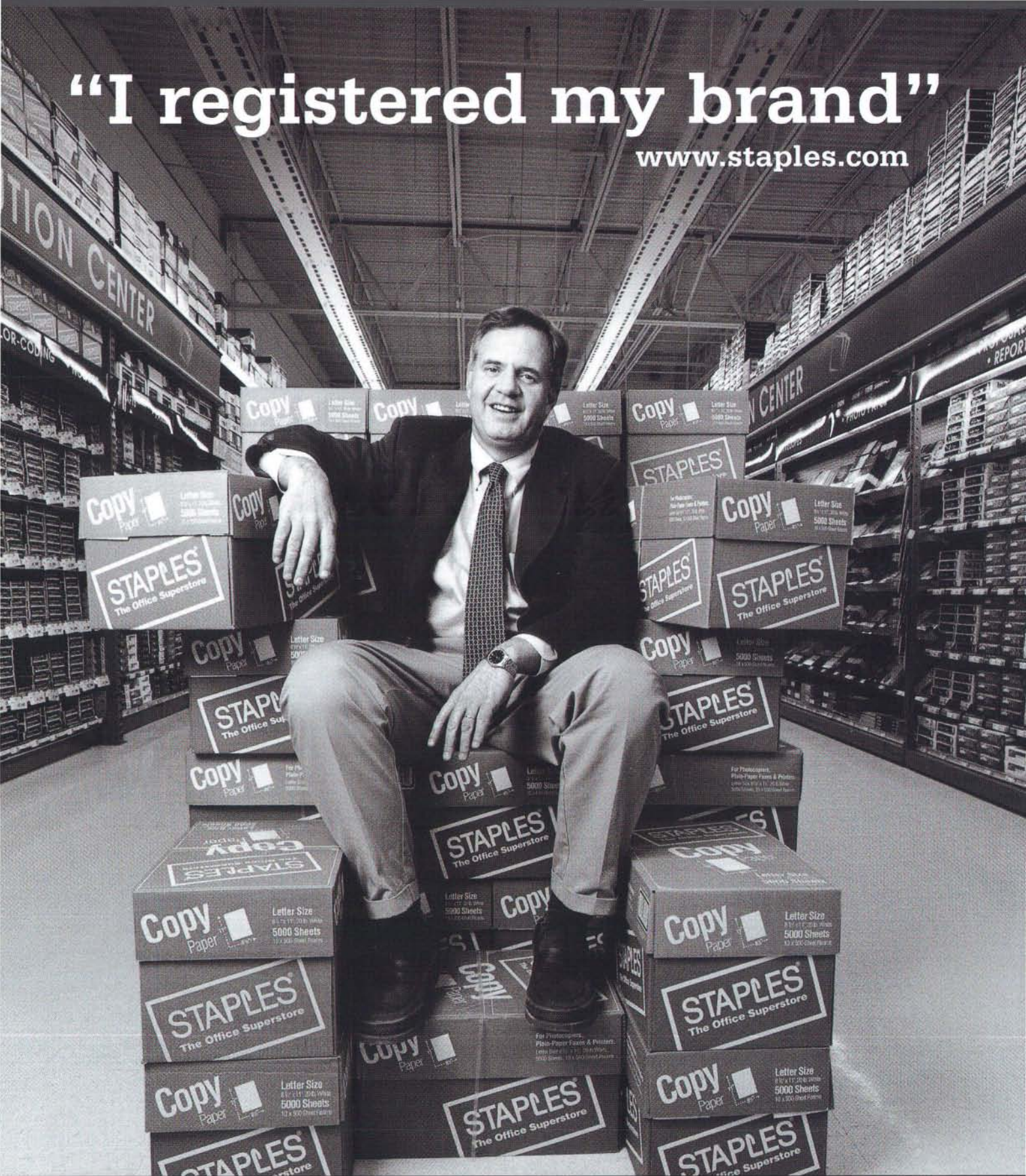
We will now show the important parts of the driver relating to the configuration dialog. For our Dynasight driver this part is relatively easy since we have only three popup menus in the dialog box (see **Figure 2**). Each menu can be used to change the function of a Dynasight axis. For example, a user might select the menu and change the function of the x-axis from need 'pitch' to need 'move forward'. The driver maintains the list of total needs of the application and a mapping of needs to axes. A selection in the dialog might change this mapping.

As was mentioned above, the first routine InputSprocket calls when starting configuration is the GetSize routine. The driver returns the minimal and optimal size we need, as shown in Listing 7. The next routine called is the GetIcon routine shown in Listing 8. The driver returns the resource identifier of the icon suite of the Dynasight driver.

Listing 7: ISpDriverDynasight.cp

GetSize

```
OSStatus GetSize(
  UInt32 refCon,
  Point *minimum,
  Point *best)
{
  refCon;
  best->h = minimum->h = 200;
  best->v = minimum->v = 120;
  return noErr;
}
```



Thomas Stemberg, CEO and founder of Staples, registered his domain name at register.com. Register your business. Register your passion. Register your dreams. Take your first step on the web today at www.register.com.

register.com™

the first step on the web℠

Listing 8: ISpDriverDynasight.cp

GetIcon

```
OSStatus GetIcon(
  UInt32 refCon,
  short *iconSuiteResourceId)
{
  refCon;
  *iconSuiteResourceId = kIconSuiteDynasight;
  return noErr;
}
```

When the Dynasight device is selected is the dialog box, InputSprocket calls the driver's Show routine listed in Listing 9. This is the time to have our dialog items added to the main dialog pane. First, some information about the dialog is stored for later use (in the Hide routine for instance). Then, the 'DITL' resource is retrieved from the resource file and appended at the end. Listing 10 shows how the reverse takes place. The Hide routine is called when another device is selected in the dialog or when the dialog is closed. Then, the driver removes its dialog items from the 'DITL'.

Listing 9: ISpDriverDynasight.cp

Show

```
OSStatus Show(
  UInt32 refCon,
  DialogPtr theDialog,
  short dialogItemNumber,
  Rect *r)
{
  OSStatus err;
  TDynasightRecPtr dyna = (TDynasightRecPtr) refCon;
  Handle ditl;

  dyna->dialogRect = *r;
  dyna->dialogPtr = theDialog;
  dyna->dialogBaseDITLCount = CountDITL(theDialog);

  // Open the resource file, get our DITL and append it
  dyna->resFileRef = FSpOpenResFile(&dyna->fileSpec,
    fsRdPerm);
  err = ResError();
  if (err != noErr) { return err; }
  ditl = GetIResource('DITL', kDITLID_Config);
  err = ResError();
  if (err != noErr) {
    CloseResFile(dyna->resFileRef);
    return err;
  }
  AppendDITL(theDialog, ditl, -dialogItemNumber);

  // Free our DITL
  ReleaseResource(ditl);

  return noErr;
}
```


Listing 10: ISpDriverDynasight.cp

Hide

```
OSStatus Hide(
  UInt32 refCon)
{
  TDynasightRecPtr dyna = (TDynasightRecPtr) refCon;
  // Restore to original count of items
  ShortenDITL(dyna->dialogPtr, CountDITL(dyna->dialogPtr) -
    dyna->dialogBaseDITLCount);
  dyna->dialogPtr = nil;
  CloseResFile(dyna->resFileRef);
  return noErr;
}
```

When the dialog is on the screen, the driver needs to respond to events to track whether there are changes in the configuration. In our case, the driver has only popup menus that it can handle by responding to mouse click events. Therefore, our driver ignores all other events that InputSprocket passes. In Listing 11, it is shown how the driver handles mouse clicks. First the position of the click is established. Then we iterate through our popup menus and call HandleAxisClick for each axis. This routine checks whether the current need of an axis changes as a result of the click.

Listing 11: ISpDriverDynasight.cp

Click

```
OSStatus Click(
  UInt32 refCon,
  const EventRecord *event)
{
  refCon;
  TDynasightRecPtr dyna = (TDynasightRecPtr) refCon;
  Point where;
  UInt32 itr;
  short itemNo;
  UInt32 oldNeed;

  where = event->where;
  GlobalToLocal(&where);

  for (itr = kDialogItem_FirstPopup;
       itr <= kDialogItem_NumPopups;
       itr++) {
    itemNo = itr + dyna->dialogBaseDITLCount;
    oldNeed = dyna->axisIndexToNeeds[itr-1];
    switch (itr) {
      case kDialogItem_XAxis:
        HandleAxisClick(dyna, where, itemNo,
          kISpElementLabel_Axis_XAxis, kAxisIndex_XAxis,
          oldNeed);
```

```
        break;
      case kDialogItem_YAxis:
        HandleAxisClick(dyna, where, itemNo,
          kISpElementLabel_Axis_YAxis, kAxisIndex_YAxis,
          oldNeed);
        break;
      case kDialogItem_ZAxis:
        HandleAxisClick(dyna, where, itemNo,
          kISpElementLabel_Axis_ZAxis, kAxisIndex_ZAxis,
          oldNeed);
        break;
      default:
        break;
    }
  }

  return noErr;
}
```

Listing 12 shows the HandleAxisClick routine. First, it is checked whether the popup is actually hit and if the need for the current axis changed as a result of the click. If the need changed, the need for that axis is first cleared. Then, the new need is activated. The current configuration is marked 'dirty' as InputSprocket requires. Finally, the popup menu is drawn with the new need setting to reflect the change in need for this axis.

Listing 12: ISpDriverDynasight.cp

HandleAxisClick

```
void HandleAxisClick(
  TDynasightRecPtr dyna,
  Point where,
  short itemNo,
  ISpElementLabel elementLabel,
  TAxisIndex axisIndex,
  UInt32 oldNeed)
{
  UInt32 itemIndex = itemNo - dyna->dialogBaseDITLCount;
  UInt32 newNeed;

  if (CheckPopUpHit(dyna, where, itemNo,
      kISpElementKind_Axis, elementLabel, validAxisKinds,
      validAxisKindsCount, oldNeed, newNeed) &&
      (oldNeed != newNeed)) {
    dynasightPtr->axisIndexToNeeds[axisIndex] = newNeed;
    dynasightPtr->configurationDirty = true;
    PlotPopupIcon(dynasightPtr,axisIndex + 1 +
      dynasightPtr->dialogBaseDITLCount, ttNone);
  }
}
```

This concludes our coverage of the driver's code for the configuration dialog. There are some parts of the driver that we could not cover such as auto-configuration. The interested reader can look at the full source code of the driver to see how this is accomplished.

Conclusion

In this month's episode, you have discovered how InputSprocket can be used to create a driver for the Dynasight device. We hope that this episode has illustrated the power of InputSprocket. To connect Cubby with the Dynasight, we could have used parts of the driver code and pasted them directly into the Cubby application code. That would have saved the time to write and debug the driver. However, now that we have a driver, we can use the same device with other InputSprocket-savvy applications (imagine looking around in a game with the Dynasight as head-tracking device...). Moreover, making Cubby support InputSprocket, we can use any 3D device with an InputSprocket driver for Cubby.

In next month's episode, we will cover the integration of Cubby with InputSprocket. We will show you how Cubby uses InputSprocket to move the cameras from part I. In addition, we will cover calibrating Cubby so that Cubby positions the cameras such that the correct images are generated.

Bibliography and References

- Apple Game Sprockets web site. The place to download Apple Game Sprockets software, documentation and example software. Available as: http://developer.apple.com/games/sprockets/index.html
- Configuring Game Input Devices with InputSprocket. Available as: http://developer.apple.com/techpubs/macos8/pdf/InputSprocket1.7.pdf.
- Djajadiningrat, J. P. & Gribnau, M. W. (1998). Desktop VR using QuickDraw 3D, Part I. MacTech Magazine 14(7), 32-43.
- Gribnau, M. W. & Djajadiningrat, J. P. (1998). Desktop VR using QuickDraw 3D, Part II. MacTech Magazine 14(8), 26-34.
- Djajadiningrat, J.P. & Gribnau, M.W. (2000). Cubby: Multiscreen Desktop VR Part I. MacTech Magazine 16(10).

PROGRAMMER'S CHALLENGE

by Bob Boonstra, Westford, MA

FreeCell

Those of you who spend time on the Dark Side might have encountered a solitaire game called FreeCell packaged with a prominent operating system by Mr. Bill. Your Challenge this month is to produce a FreeCell player.

The prototype for the code you should write is:

```
typedef enum {
  kNoSuit=0, kSpade, kHeart, kDiamond, kClub
} Suit;

typedef enum {
  kNoSpot=0,
  kAce, k2, k3, k4, k5, k6, k7, k8, k9, k10,
  kJack, kQueen, kKing
} Spot;

typedef struct Card {
  Suit suit;  /* the suit of the card, kSpade .. kClub */
  Spot spot;  /* the spot of the card, kAce .. kKing */
} Card;

typedef enum {  /* places to move cards from */
  sFreeCellA=2,sFreeCellB,sFreeCellC,sFreeCellD,

sTableau0=6,sTableau1,sTableau2,sTableau3,sTableau4,sTableau5,s
Tableau6,sTableau7
} Source;

typedef enum {  /* places to move cards to */
  dHome=1,
  dFreeCellA=2,dFreeCellB,dFreeCellC,dFreeCellD,
  dTableau0=6,dTableau1,dTableau2,dTableau3,
  dTableau4,dTableau5,dTableau6,dTableau7
} Destination;

typedef struct Move {  /* move a card from theSource to theDestination */
  Source theSource;
  Destination theDestination;
} Move;

typedef struct Tableau {  /* each Tableau can contain 0..13 cards */
  Card theCard[13];
} Tableau;

long /* numMoves */ FreeCell(
  const Tableau theTableau[8],  /* the cards as initially dealt */
  Move theMoves[],              /* return your moves in order here */
  long maxMoves                 /* storage is preallocated for maxMoves theMoves */
);
```

The FreeCell game is different from many solitaire games in a couple of respects. First, all of the cards are visible, so winning the game is more a matter of strategy than of luck. Second, while there are FreeCell deals that cannot be solved, nearly every game can be won, as contrasted with less than half of other popular solitaire games.

FreeCell starts with the playing Cards dealt face up into 8 piles called Tableaus. All 52 Cards are used, which means that the first four Tableaus receive seven Cards, and the remaining four Tableaus receive six Cards. The object of the game is to move all of the Cards onto four "Home" piles, one for each Suit, played in order from Ace up to King. You also have available four temporary locations, or "Free Cells", each of which can hold one Card. A Move consists of one of the following:

- moving an Ace from a Free Cell or from the top of a Tableau to an empty Home pile
- moving the next higher Card of a Suit from a Free Cell or from the top of a Tableau to the Home pile for that Suit
- moving a Card from the top of a Tableau to an empty Free Cell
- moving a Card from the top of a Tableau or a Free Cell to an empty Tableau
- moving a Card from the top of a Tableau or from a Free Cell to the top of a Tableau, where the Suit of the Card on top of the destination Tableau has the opposite color of the Card being moved, and where the Spot of the Card on top of the destination Tableau is one higher than the Spot of the Card being moved.

Cards can be moved to or from a Free Cell, but each Free Cell can hold only one Card. Cards can be moved to the Home piles, never back from the Home piles. Cards can be moved to or from the top of a Tableau, but they can only be moved to a Tableau if the Suit colors alternate and if the Card value (Spot) decreases by one. Any Card from a Free Cell or the top of another Tableau may be placed on an empty Tableau.

Your FreeCell routine will be called with the Cards dealt into the 8 Tableaus. Your task is to generate a sequence of Moves that solve the puzzle, returning them in theMoves. Each Move consists of a Source and a Destination. It is not necessary to specify the Card being moved, because the Source uniquely identifies the Card at any given point in the game. FreeCell should return the number of Moves generated, or zero if you are unable to solve the puzzle.

Your solution will be awarded 10,000 points for each puzzle it solves correctly, and penalized 1 point for each millisecond of processor time required to solve the puzzle.

This will be a native PowerPC Challenge, using the CodeWarrior Pro 5 environment. Solutions may be coded in C, C++, or Pascal.

This Challenge was suggested by Peter Lewis, who wins 2 Challenge points for the suggestion. More information on the game FreeCell can be found at http://www.freecell.org.

Mac Support Since 1985!

New Version! Enhanced Security Options, New Platforms, More...

c-tree Plus®

ONE EMBEDDED DATABASE TOOL THAT FITS ALL YOUR APPLICATIONS

FairCom has been providing fast, flexible and scalable database development tools to the commercial developer for over 20 years. During this time FairCom has been utilized within countless embedded appliances, web server development projects and many vertical market applications. By offering industry leading performance, unsurpassed multi-user data availability and a complete transaction enabled database Server, FairCom provides the depth and breadth of technology to fit all of your database development needs. Add FairCom to your toolbox today.

ONE FAST ISAM TOOL, MANY PLATFORMS

Every copy of c-tree Plus supports all these platforms: Mac OS, Mac OS X, MK Linux, Linux (Intel...), Novell Netware, OS/2, Solaris (SPARC), Solaris (Intel), Sun OS, AIX, HP UX, SCO, Interactive, AT&T Sys V, QNX, 88OPEN, FreeBSD, Windows 95/98/2000/NT, Lynx, Banyan Vines, and more...

Plus, support for ADSP, SPX, TCP/IP

Embedded Hardware

APPLICATION	BENEFIT
Internet Appliances	Small footprint
Medical Devices	Stable
Factory Automation	Includes Full Source Code
Space Exploration	Proven Technology

Web Development

APPLICATION	BENEFIT
Dedicated Web Server	Robust Threading
B2B Server	Flexible Communication

Vertical Markets

APPLICATION	BENEFIT
Library Management	Fast
Insurance	Reliable
Inventory Control	Scalable
Point of Sale	Cross Platform

c-tree Plus®

- *Industry leading performance*
- *Seamless cross platform migration*
- *Unsurpassed multi-user data availability*
- *Complete source code*
- *Portable threading API*
- *Thread safe libraries*
- *Low total cost of ownership!*

FairCom® Server

- *Powerful multithreaded database server*
- *Available for over 25 platforms*
- *Robust heterogenous network support*
  - Mix and match clients and servers
- *Multithreaded client support*
- *Full transaction processing*
- *Automatic file recovery*
- *Very small RAM and disk footprint*
- *Easy installation/configuration*
  - No DBA required
- *Much more!*

...IN ONE CONVENIENT PACKAGE FOR ONLY $895

FairCom Offices

USA	573.445.6833
EUROPE	+39.035.773.464
JAPAN	+81.59.229.7504
BRAZIL	+55.11.3872.9802

FairCom® CORPORATION

Commercial Database Solutions Since 1979

www.faircom.com/mac • USA. 800.234.8180 • info@faircom.com

Three Months Ago Winner

Congratulations once again to **Ernst Munter (Kanata, Ontario)** for submitting the winning entry to the August Longest Word Sort Challenge. Ernst's entry was the fastest of the seven entries submitted, and was just under 40% faster than the second-place entry by Jonny Taylor.

The Longest Word Sort Challenge required contestants to sort a sequence of lines of text based on the length of words in each line. The line with the longest word was to be considered greater than any other line. Among lines with longest words of the same length, the comparison was to be based on the next longest word, etc. Among lines with words of exactly the same length, the order was to be based on an alphanumeric comparison of words, in order of length, and then in order of occurrence.

The key to Ernst's solution is the LineDescriptor that he uses to profile each line of text. The LineDescriptor contains a packed description of the number of words of each length contained in the line. This allows Ernst to compare lines by comparing the numeric line profile values, using a single subtraction in the LineDescriptor::IsLessThan routine to compare the number of words of several lengths. In the event the LineDescriptors match exactly, the CompText routine compares the words of each line as text, in order of word length. This Challenge allowed the use of assembly language, and Ernst used one line of it in the BitsNeeded routine, which is used by Text::ComputeFieldSizes to calculate the width of the packed field needed to hold the number of words of a particular length.

Jonny Taylor's second-place solution uses a combination of sorting techniques, starting with a radix sort to partially sort the list based on the lengths of the 16 longest words in each line, and then using a quicksort algorithm to sort groups of lines with identical word lengths. Jan Schotsman's third place solution uses a merge sort to compare groups of up to 32 lines, starting with a comparison of the lengths of the 16 longest words in each line, and resorting to a more careful comparison when necessary. This Challenge certainly produces an interesting variety of approaches to an unusual sorting problem.

The table below lists, for each of the solutions submitted, the cumulative execution time in milliseconds. It also provides the code size, data size, and programming language used for each entry. As usual, the number in parentheses after the entrant's name is the total number of Challenge points earned in all Challenges prior to this one.

Name	Time (msecs)	Code Size	Data Size	Lang
Ernst Munter (631)	168	3384	330	C++
Jonny Taylor (26)	272	14068	44	C
Jan E. Schotsman	337	7256	56	C
Rob Shearer (47)	417	42616	965	C++
Darrell Walisser	630	4124	128	C
Ladislav Hala (7)	638	3128	2429	C
Ron Nepsund (47)	972	6492	501	C

Top Contestants ...

Listed here are the Top Contestants for the Programmer's Challenge, including everyone who has accumulated 20 or more points during the past two years. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	Munter, Ernst	243	5.	Shearer, Rob	51
2.	Saxton, Tom	106	6.	Boring, Randy	50
3.	Maurer, Sebastian	68	7.	Taylor, Jonathan	36
4.	Rieken, Willeke	65	8.	Brown, Pat	20

... and the Top Contestants Awaiting Their First Win

Starting this month, in order to give some recognition to other participants in the Challenge, we are also going to list the high scores for contestants who have accumulated points without taking first place in a Challenge. Listed here are all of those contestants who have accumulated 6 or more points during the past two years.

Rank	Name	Points	Rank	Name	Points
9.	Downs, Andrew	12	17.	Wihlborg, Claes	9
10.	Jones, Dennis	12	18.	Hala, Ladislav	7
12.	Duga, Brady	10	20.	Schotsman, Jan	7
13.	Fazekas, Miklos	10	21.	Widyatama, Yudhi	7
15.	Selengut, Jared	10	22.	Heithcock, JG	6
16.	Strout, Joe	10			

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place	20 points
2nd place	10 points
3rd place	7 points
4th place	4 points
5th place	2 points
finding bug	2 points
suggesting Challenge	2 points

Here is Ernst's winning Longest Word Sort solution:

LongestWordSort.
cp


```
/*

The Problem
-----------
Text is to be sorted by lines, with the lengths of words as well as the text itself
determining the order of lines.
```

Solution

The text is analyzed, and a line descriptor derived for each line. The line descriptor contains a profile which lists the frequency of words by length for the referenced line. This profile is the primary key for the sort. Text comparisons only come into play when the profiles are identical.

To reduce the main sort effort, lines are pre-sorted into groups which share the same longest word length.

Each group is then sorted with heap sort, where the first phase (of inserting the line into a heap) is combined with making a copy of the line of text into a temporary pre-sorted text, i.e. by line group. The second phase of the heap sort is then combined with copying the result back to the external text array, in the final order.

Optimizations

Memory accesses are minimized by copying the text just once to temporary storage and back, as part of the sorting opration.

The sorting itself is conducted with short line descriptors, based on bit-packed profiles. In most cases, a line descriptor will be 16 or 20 bytes long.

The number of comparisons is reduced (further economizing on memory access) by using a combination of distribution sort (by line group) and heap sort within each line group segment. Null and null-word lines remain in the original order and are already sorted by the distribution sort.

Most functions are written as inlined, but the compiler will not necessarily inline them, using its own "smart"ness. This is not necessarily optimal. I have forced it not to inline Pop() and CompText(). This allows the compiler then to inline other function, reducing the number of function calls overall.

Assumptions

TextToSort should end with a Mac newline character (0x0d); Any text beyond the last newline character will not be sorted.

No assumptions are made for limits except that the longest word must be less than 128 characters long. Lines may be of any length and contain any number of words.
*/

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "LongestWordSort.h"

#define MAXLENGTH 127  // no word longer than MAXLENGTH chars is expected

typedef unsigned char uchar;
typedef unsigned short ushort;
typedef unsigned long ulong;

enum {
   kMaxLength=MAXLENGTH,
   kArraySize=1+MAXLENGTH,
   kEOL=0x0d,
   kAlnumType=0,
   kWhiteType=1,
   kEOLtype=2,
   kCaseSensitive=0xFF,
   kNoCase=0xDF,
   kSignBit=0x80000000
};

static uchar kCaseMask[2] = {kNoCase,kCaseSensitive};
static ulong gCaseMask;
static ulong gDescending;

inline long Max(long a,long b)
// Algebraic method of selecting greater of two longs, avoids branch stalls
{
   long D = (a-b) >> 31;
   long notD = ~D;
   return (b & D) | (a & notD);
}
```

BitsNeeded

```
inline long BitsNeeded(ulong x)
// Returns number of bits needed to encode range 0 to x
{
   // this simple intrinsic is just what the job needs
   return 32 - __cntlzw(x);
/*
   // platform independent method:
   int n=0;
   while(x) { x >>= 1; n++;}
   return n;
*/
}

// Private version of the table from ctype, to include a code for EndOfLine (kEOL)
static struct CharType {
   uchar T[256];
   CharType()
   {
      for (long c=0;c<256;c++)
         T[c]=(isalnum(c))?kAlnumType:kWhiteType;
      T[kEOL]=kEOLtype;
   }
} gCharType;
```

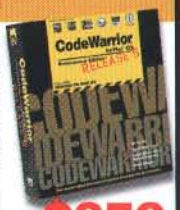
struct LineDescriptor

```
// A LineDescriptor characterises one line of text,
// containing all information needed for efficient sorting
struct LineDescriptor;
typedef LineDescriptor* LineDescriptorPtr;
struct LineDescriptor {
   uchar* textRef;        // pointer to start of line in copy of text
   ulong  lineLength;     // number of chars of this line
   ushort longestWordLength; // length of the longest word in this line
   ushort profileLength;  // number of longs in profile
   long profile[1];       // packed, number of words per length

   ulong LineLength() const {return lineLength;}

   long Init(uchar* lineStart,ulong lineLen,ulong longest,
      ulong lengthDist[],uchar fieldWidth[])
```

The products you need, with the prices and service you deserve... guaranteed.

Power Tools for Programmers!

CodeWarrior Pro 5



CodeWarrior Professional 5 put everything you need for software development at your fingertips: project management tools, text and resource editors, source and class browsers, compilers, linkers, assemblers, and debugger.Release 5 offers such features as RAD for Java, faster compile times, local and remote application debugging, IDE extensibility options and even tighter C/C++ compliance. Additionally, you can create applications for Windows 95/98/NT and Mac OS 8.x and Mac OS X from either host platform. (available in versions hosted on Mac or Windows).

$359

Installer Maker 6.5 10K

Whether you're a developer of high-end, complex applications, simpler utilities, shareware/freeware, an IS manager or an ISP who needs to distribute files quickly and easily, the new InstallerMaker is the complete installation solution for you. Utilizing the power of the new StuffIt Engine, InstallerMaker creates installers faster and smaller than ever, decreasing download time off servers and reducing the number of disks needed to distribute installers. These time and cost savings go straight to your bottom line!

$219

Spotlight

Spotlight is the first Macintosh "Automatic Debugger". It can automatically locate run time errors in your code and display the offending source code line. Unlike similar tools on other platforms Spotlight is easy to use. No source code changes are necessary for application debugging. Spotlight can automatically check for wild pointers, memory leaks, overwrites, underwrites, invalid dereferencing of handles, and even toolbox parameter validity checking -- spotlight knows Macintosh verifying parameters to over 400 toolbox calls.

$189

VOODOO Server

VOODOO Server is a version control system for software developers using Metrowerks CodeWarrior under Mac OS. VOODOO Server and the corresponding VOODOO clients (the included CodeWarrior VCS plug-in and the VOODOO Admin application) are designed to offer reliable and robust version control features while minimizing the administrative overhead that usually accompanies version control. If you're a single programmer or managing a team of developers, version control can make or break your project. Do it right, with VOODOO

$79

Resorcerer 2.2

Resorcerer is the only supported general-purpose resource editor for Macintosh. Relied upon by thousands of Mac developers, Resorcerer features a wealth of powerful yet easy-to-use tools for easier, faster, and safer editing of Macintosh data files and resources. Whether you have to parse a picture, debug a data fork, design and try out Balloon Help, create a scripting dictionary, create anti-aliased icons, design and edit a custom resource with 40,000 fields in it, create C source code to run a dialog, or any of hundreds of other resource-related tasks, Resorcerer's magic will quickly save you time and money.

$256

Future BASIC 3

NEW!

One of the most flexible and powerful development environments on the Macintosh today! Easily create programs with the visual program editor, drop into the BASIC editor to define powerful logic with the worlds easiest programming language, or work directly with the Macintosh toolbox. The only BASIC compiler on the market that gives you 100% access to all of the power of the Macintosh toolbox!

$159

and hundreds more!

Page Charmer 2.0 $139

Scripter 2.0 $179

WebTen $349

ToolsPlus Lite $99

FaceSpan 3.0 $179

WebSpice 1,000,000 $89

WebSpice Animations $89

MkLinux $39

PowerKey Rebound $89

Developer DEPOT®

PO Box 5200 • Westlake Village, CA • 91359-5200 • Voice: 800/MACDEV-1 (800/622-3381)
Outside US/Canada: 805/494-9797 • Fax: 805/494-9798 • E-mail: orders@devdepot.com

www.devdepot.com

Master the Web!

WebSTAR Server Suite 4.2

WebSTAR Server Suite is a complete set of powerful and easy-to-use Internet servers for the Mac OS. Effortlessly serve web pages, host email accounts, publish databases, and share files - all with a single application on one Mac! WebSTAR Server Suite is perfect for Internet or Intranet serving, single or multiple sites, smalll and large businesses - it's power and ease-of-use saves any organization time and money.

$539

CyberGauge 3.0

NEW!

Monitor the bandwidth usage of up to five different machines on your network! Do you need to upgrade your webserver? How hard is your eMail server working? Are you getting all the bandwidth you're paying for? Not only can CyberGauge answer all these questions, new features allow CyberGauge to eMail or page your network device becomes unresponsive r passes a threshold of usage you define – an essential first line of defense for early detection of denial of service attacks and necessity for warning you and tracking quality of ISPs that may have brown outs and shutdowns.

$249

Funnel Web Pro

NEW!

Funnel Web is the ultimate web analysis solution for professionals. Specifically designed for profiling web site usage and monitoring customer usage patterns, Funnel Web is ideal for examining server performance and online effectiveness. Funnel Web can analyze log file formats from any server, WebSTAR, WebTen, even Unix or NT hosted ervers. Discover the most popular pages on your site, track server loads & optimize server performance, profile visitors based on organization, domain name, country, browser, etc. Funnel Web does it all!

$329

NetBarrier

NetBarrier offers a Personal Firewall, Antivandal protection, and Internet Filtering. Protect your machine from intrusions by Internet or AppleTalk. Incorrect passwords and individual actions are logged, you are alerted to hostile actions, and intruders are easily isolated. Internet Filtering allows you to be sure that passwords, credit card numbers, and other sensitive information can never be exported from your computer -- the content itself is filtered before any transfer! (Rated 4 mice by Macworld Magazine)

$57.95

...and great hardware solutions!

2 USB PCI Card

Add two USB ports to your older Macintosh. Connect up to 127 devices to the Universal Serial Bus (USB) that is Apple's new standard for desktop connectivity. USB mouse devices, keyboards, joysticks, game controllers, printers, scanners — connect them all to your current computer. Installs in minutes!

$32.95

Dr. Bott Moni Switch ADB or USB

NEW!

Do you need 4 monitors and 4 keyboards for your 4 servers? With Dr. Bott Moni-Switch you can connect a single keyboard and monitor to up to 4 machines at once! A simple flick of a switch directs the video input and keyboard commands to the appropriate CPU! Available in USB and ADB models, with 2 or 4 machine support, and bundles with USB PCI cards so you can mix and match USB and ADB machines with the same Moni-Switch! Great for programmers to do back ground compiles, ideal for server rooms overcrowded with monitors and keyboards!

As low as

$129.95

Macsense USB Full Size Keyboard

Just plug this keyboard into your Mac and start typing! The UKB-600 keyboard from Macsense is designed to get you typing quickly and easily, without any hassle or compatibility worries. It features two tone translucent design, colored to match your favorite flavor of Macintosh. It offers soft touch with positive tactile feedback and build built in USB port on either side of the keyboard. Includes a 5' USB cable and is 100% Macintosh compatible, simply plug and play, as easy as Macintosh!

$44.95

Macsense Internet Sharing Router

Looking to get your whole office online without shelling out thousands of dollars? If so, the XRouter Internet Sharing Hub offers the perfect solution. This amazing Ethernet-to-Ethernet hub connects an entire network of up to 252 users to the Internet using only one ISP account and one Cable or DSL modem!

$199

```
// Initializes the fields for one line.
// Returns the length of the struct, which includes the variable-size profile
  {
    textRef=lineStart;
    lineLength=lineLen;
    longestWordLength=longest;
// Builds the line profile by packing the number of words, by length,
// into a string of 31-bit values for efficient comparison later.
    long* prof=profile;
    ulong acc=0;
    ulong fill=0;
    for (long len=longest;len>0;--len)
    {
      long numBits=fieldWidth[len];
      long value=lengthDist[len];
      fill+=numBits;
      if (fill > 31)
      {
        fill=numBits;
        *prof++=acc;
        acc=value;
      } else
      {
        acc = value | (acc << numBits);
      }
      lengthDist[len]=0;
    }
    *prof++=acc;
    profileLength=prof-profile;
    return (sizeof(LineDescriptor) - sizeof(ulong)) +
      sizeof(ulong) * profileLength;
  }
```

LineDescriptor::IsLessThan

```
  ulong IsLessThan(const LineDescriptor& other) const
// Returns true if this line should be "before" the other line (ascending)
  {
// Compare line profiles first:
    const long* A=profile;
    const long* B=other.profile;
    long pLength=profileLength;

    // there is always at least one profile word to compare
    long delta = (*A - *B);
    if (delta)
      return delta & kSignBit;

    while (--pLength) {
      delta = *++A - *++B;
      if (delta)
        return delta & kSignBit;
    }

// Both profiles are equal: must compare on the basis of text:
    delta=CompText(other);
    if (delta)
      return  delta & kSignBit;

// The text is exactly the same:
// Compare on the basis of the original line order
// But reverse if descending to compensate for reverse CopyBack
    return gDescending ^ ((textRef - other.textRef) &
kSignBit);
  }

  static long CompWords(const uchar* w1,
    const uchar* w2,long len,long caseMask)
// Simple string comparison, character by character, case sensitive as required
  {
    for (long i=0;i<len;i++)
    {
      long c1=*w1++ & caseMask;
      long c2=*w2++ & caseMask;
      long delta=c1-c2;
      if (delta) return delta;
    }
    return 0;
  }
```

LineDescriptor::LocateNext

```
static const uchar* LocateNext(const uchar* start,long length)
// Returns the next word of the specified length
  {
    ulong c=*start,len=0;
    const uchar* word=start;
    for (;;)
    {
      long charType=gCharType.T[c];
      c=*++start;
      if (charType==kAlnumType)   // is alnum
      {
        if (len==0) word=start-1;
        len++;
      } else
      {
        if (len==length)      // lengths match: found it
            break;

        if (charType==kEOLtype)  // reached end of the line
          return 0;
        len=0;
      }
    }
    return word;
  }

  void Write(uchar* outText) const
  {
// Note: memcpy is faster than strncpy or character by character copy.
    memcpy(outText,textRef,lineLength);
  }

  long CompText(const LineDescriptor& other) const;
};
```

LineDescriptor::CompText

```
long LineDescriptor::CompText(const LineDescriptor& other)
const
// Returns result of comparing the line as text, longest words first
{
  for (long len=longestWordLength;len>0;len--)
  {
    const uchar* w1=textRef;
    const uchar* w2=other.textRef;
    for (;;)
    {
      if (0==(w1=LocateNext(w1,len)))
        break; // no word of this length found

      w2=other.LocateNext(w2,len);
      // if w1 exists, w2 must exist
      long d=CompWords(w1,w2,len,gCaseMask);

      if (d)
        return (d);
      w1 = w1+len;
      w2 = w2+len;
    }
  }
  return 0;
}
```

struct Segment

```
// The priority queue (Heap) structure is used for sorting the line descriptors.
// Sorting occurs in two phases: Insert() and Pop()
// We build a separate heap for each lineGroup segment (which shares a common
// longest word length). This reduces the size of the individual heaps, resulting in
// fewer comparisons overall.
struct Segment {
  LineDescriptorPtr*   heapBase;
  ulong  heapSize;
  ulong  maxHeapSize;
  uchar* nextLineDesc;
  ulong  lineDescSize;

// Just keep track of the line count, to prepare correct heap size

  void AddLine(){maxHeapSize++;}

  ulong MemRequired(ulong profileLongs,ulong profileBits)
  {
// Returns the amount of memory required for lineDescs and their index (=heap)
    if (maxHeapSize==0) return 0;
    ulong profileSizeInLongs=profileLongs+(31+profileBits)/32;
    ulong profileBytes=4*profileSizeInLongs;
```

```
    lineDescSize=
      profileBytes + (sizeof(LineDescriptor) - sizeof(ulong));
    return
      lineDescSize * maxHeapSize +          // for linedescs
      sizeof(LineDescriptorPtr*) * (1+maxHeapSize);// for heap
index
  }

  uchar* Init(uchar* memPool)
  {
// Grabs memory from the pool for lineDescs and the heap; returns updated pool
    if (maxHeapSize>0)
    {
      nextLineDesc=memPool;
      memPool += maxHeapSize * lineDescSize;

      heapBase=(LineDescriptorPtr*) memPool;
      memPool += sizeof(LineDescriptorPtr*) * (1+maxHeapSize);
    }
    return memPool;
  }

  LineDescriptorPtr GetLineDesc()
// Walks the index memory to assign consecutive lineDesc indices
  {
    LineDescriptorPtr next=LineDescriptorPtr(nextLineDesc);
    nextLineDesc += lineDescSize;
    return next;
  }

  void Insert(LineDescriptorPtr k)
// Inserts one line, while maintaining the heap property
  {
    long i=++heapSize;
    long j=i>>1;
    LineDescriptorPtr z;
    while (j && (k->IsLessThan(*(z=heapBase[j]))))
    {
        heapBase[i]=z;
          i=j;
        j=i>>1;
    }
    heapBase[i]=k;
  }

  uchar* CopyBack(uchar* textToSort)
// Pops one line off the heap, and copies the referenced line to the output.
// When descending, lines are copied starting at the end of the output.
// Returns the updated output text pointer, in preparation for the next copy.
  {
    if (0==heapSize)
      return textToSort;

    if (gDescending)
    {
      uchar* endText=textToSort;
      do {
        LineDescriptorPtr textLine=Pop();
        uchar* outText=endText - textLine->LineLength();
        textLine->Write(outText);
        endText=outText;
      } while(heapSize);
      return endText;
    } else
    {
      uchar* outText=textToSort;
      do {
        LineDescriptorPtr textLine=Pop();
        textLine->Write(outText);
        outText+=textLine->LineLength();
      } while(heapSize);
      return outText;
    }
  }

  LineDescriptorPtr Pop();
};
```

Segment::Pop

```
LineDescriptorPtr Segment::Pop()
// The node at heapBase[1] has the lowest weight.
// It is popped from the heap and returned.
// The heap is adjusted to maintain the heap property.
{
  LineDescriptorPtr rc=heapBase[1];
    LineDescriptorPtr k=heapBase[heapSize--];
    if (heapSize<=1) {
      heapBase[1]=k;
      return rc;
    }
    long i=1,j=2;
    while (j<=heapSize)
    {
      if ((j<heapSize)
    && (heapBase[j+1]->IsLessThan(*heapBase[j])))
        j++;
      if (k->IsLessThan(*heapBase[j]))
        break;
      heapBase[i]=heapBase[j];
      i=j;j+=j;
    }
    heapBase[i]=k;
    return rc;
}
```

struct Text

```
struct Text {
  uchar*   theText;
  ulong  textSize;
  uchar* copyText;
  ulong  gNumLines;
  ulong  gLongest;

  uchar* lineDescMemory;

// highest number of words of length x in a line:
  ulong  lengthDist[kArraySize];
  ulong  tempLengthDist[kArraySize];

// one segment of lineDescriptors for each possible lineGroup
  Segment  segment[kArraySize];
```

```
// size (=numChars) of each text segment by longestWord line group:
  ulong  segmentSize[kArraySize];

// start of each text segment by longestWord line group
  uchar* segmentStart[kArraySize];

// width of profile fields in bits for each possible word length
  uchar  fieldWidth[kArraySize];

  Text(char *textToSort,long numChars) :
       theText((uchar*)textToSort),
       textSize(numChars),
       copyText(new uchar[numChars])
  {
// Constructor analyzes text and computes text profile, prior to sorting.
// Determines the division of the text into line groups.
    memset(lengthDist,0,sizeof(lengthDist)
      + sizeof(tempLengthDist)
      + sizeof(segment)
      + sizeof(segmentSize));
    Analyze();
    ComputeFieldSizes();
    lineDescMemory=GetLineDescMemory();
  }
  ~Text()
  {
    delete [] lineDescMemory;
    delete [] copyText;
  }

  void Sort()
// Does the actual sorting of the segments:
  {
// Scans the text a second time while inserting lines in segments heaps.
// This constitutes the first phase of sorting.
    Presort();

// Zero-word lines will be in the original order in line group 0,
// They can just be copied en bloc, to the front or back end of the output.
    uchar* dest=theText;
    if (gDescending)
    {
      dest += textSize-segmentSize[0];
      memcpy(dest,copyText,segmentSize[0]);
    } else
    {
      memcpy(dest,copyText,segmentSize[0]);
      dest += segmentSize[0];
    }

// The remaining lines are popped from each segment heap,
// starting with the linegroup with the shortest longest words.
// This completes the second phase of sorting.
    for (long len=1;len<=gLongest;len++)
    {
      dest=segment[len].CopyBack(dest);
    }
  }
  void Analyze();
  void ComputeFieldSizes();
  uchar* GetLineDescMemory();
  void Presort();
};
```

Text::Analyze

```
void Text::Analyze()
// Scans the original text and collects statistics such as
//    - the number of lines
//    - the number of words by length
//    - the size of each line group (lines with the same max-length)
//    - the longest word in the whole text
// The loop in this routine relies on finding a 0x0d character at text end.
{
  ulong  len=0,longest=0,numLines=0;
  ulong  globalLongest=0;
  uchar* text=theText;
  ulong  c=*text;
  uchar* lineStart=text;
  ulong  numChars=textSize;
  for (;;)
  {
    long charType=gCharType.T[c];
    c=*++text;
    if (charType==kAlnumType) len++;   // part of a word
    else
    {
      if (len)                 // register the word
      {
        tempLengthDist[len]++;
        longest=Max(longest,len);
        len=0;
      }

      if (charType==kEOLtype)          // register the line
      {
        for (long l=0;l<=longest;l++)
        {
          if (lengthDist[l] < tempLengthDist[l])
            lengthDist[l] = tempLengthDist[l];
          tempLengthDist[l]=0;
        }
        segment[longest].AddLine();
        globalLongest=Max(globalLongest,longest);
        ulong lineLength=text-lineStart;
        segmentSize[longest]+=lineLength;
        numChars -= lineLength;
        numLines++;

        if (numChars <= 0)
          break;
        longest=0;
        lineStart=text;
      }
    }
  }
  gNumLines = numLines;
  gLongest  = globalLongest;
}
```

Text::ComputeFieldSizes

```
void Text::ComputeFieldSizes()
// Computes the minimum field widths for profiles, for each word length
// also devides the text copy into segments, one per line group
{
  uchar* start=copyText;
  for (long len=0;len<=gLongest;len++)
  {
    fieldWidth[len] = BitsNeeded(lengthDist[len]);
    segmentStart[len]=start;
    start+=segmentSize[len];
  }
}
```

Text::GetLineDescMemory

```
uchar* Text::GetLineDescMemory()
// Allocates the memory required for the variable size line descriptors
// and sets up the lineGroup segments.
// Note: In the interest of not fragmenting the memory heap unnecesssarily,
//       this memory is allocated as a single chunk, and then divided
//       out among the segments for line descriptors and index arrays.
{
  ulong memRequired=0;

  ulong profileBits=0,profileLongs=0;
  for (long len=1;len<=gLongest;len++)
  {
    long fWidth=fieldWidth[len];
    if (fWidth)
    {
      // accumulate total bits to cover fields up to length len
      profileBits += fWidth;
      if (profileBits > 31)
      {
        profileLongs++;
        profileBits=fWidth;
      }

      // if segment[len] is not empty, reserve memory for it
      memRequired +=
          segment[len].MemRequired(profileLongs,profileBits);
    }
  }

// Allocate all required memory ..
  uchar* allocated=new uchar[memRequired];
  memset(allocated,0,memRequired);

// .. and divide it among active segments
  uchar* memPool=allocated;
```

```
  for (long len=1;len<=gLongest;len++)
    memPool=segment[len].Init(memPool);

  return allocated;
}
```

Text::Presort

```
void Text::Presort()
// Second scan of the text:
//     assigns and initializes a line descriptor for each line,
//     and inserts it in the selected segment
{
  uchar* text=theText;
  ulong len=0,longest=0;
  memset(tempLengthDist+1,0,gLongest*sizeof(ulong));
  uchar* lineStart=text;
  ulong c=*text;
  long numLines=gNumLines;
  for (;;)
  {
    long charType=gCharType.T[c];
    c=*++text;
    if (charType==kAlnumType) len++;
    else
    {
      if (len)          // register the word
      {
        tempLengthDist[len]++;
        longest=Max(longest,len);
        len=0;
      }

      if (charType==kEOLtype)      // register the line
      {
        --numLines;
        ulong lineLength=text-lineStart;

        uchar* copyDest=segmentStart[longest];
        memcpy(copyDest,lineStart,lineLength);
        segmentStart[longest] = copyDest+lineLength;
        lineStart=text;

        if (longest)        // make a new line descriptor
        {
          LineDescriptorPtr lineDesc =
            segment[longest].GetLineDesc();

          // note side effect: Init clears tempLengthDist
          lineDesc->Init(copyDest,lineLength,longest,
            tempLengthDist,fieldWidth);

          segment[longest].Insert(lineDesc);

          longest=0;
        } // else, no need to do anything (no words in line)
        if (numLines<=0)
          break;
      }
    }
  }
}
```

LongestWordSort

```
void LongestWordSort(
    char *textToSort,        /* the text to be sorted */
    long numChars,           /* the length of the text in bytes */
    Boolean descending,      /* sort in descending order if true */
    Boolean caseSensitive    /* sort is case sensitive if true */
) {
// Just to be sure, let's ignore all text beyond the last CR
  while (numChars && (kEOL != textToSort[numChars-1]))
    numChars--;

  if (numChars==0) return; // quit if there is no text to sort

// Make sort parameters global
  gDescending=(descending)?kSignBit:0;
    gCaseMask=kCaseMask[caseSensitive];

// Initialize ..
  Text* T=new Text(textToSort,numChars);
// and sort:
  T->Sort();
  delete T;
}
```

MT

By John C. Welch
Edited by Ilene M. Hoffman

Mac OS X Public Beta

An Administrator's Review

Welcome

Before I review Mac OS X, the next generation of operating system from Apple Computer, Inc., I'd like to emphasize one point: **It's A Beta**. The fact that it's called a public beta should make that emphasis unnecessary, but some comments I've read on the Internet and mailing lists makes me think that the emphasis is needed. This is a review of the Mac OS X Public Beta, which means that a lot of what I dislike or like may change, so don't think that anything I mention will be included in the final product. Finally, I've only had the public beta for about a week and a half at this writing, so there are a lot of items I may not cover yet. Okay, enough warning, on to the Beta!

Installation

The first place to start is the installation of Mac OS X Public Beta. I have been running the Public Beta on a PowerBook G3 Series, Bronze Keyboard, 1999. It has 192MB of RAM, and a third-party 18GB hard disk with two partitions, the Beta being on the second partition, which is about 3GB in size. I have not been using Classic Compatibility Environment for two reasons: First of all, I have certain extensions and configurations that aren't compatible with Classic that I need to use in my daily work. Secondly, I wanted to get a feel for OS X as its own operating system, without falling back on Classic as a safety net.

After reading the Read Me files, installation notes, and other information available on Apple's website (http://www.apple.com/osx/), I booted from the Beta CD, and started the install process. When you boot from the CD, you boot into the OS X install program. Here is one of my first beta gripes, as a network administrator, one of the things I really like about the Mac OS over Windows NT/2000, is that when I boot from a CD, I boot into the Mac OS, with basic networking capability enabled. can boot from the CD, get onto my network, and have full access to utilities, install points, etc. Currently the Mac OS X Public Beta CD only boots into the installer, which is annoying, and hopefully is not the future design of the final product. Aside from that, the install is fairly uneventful.

> Note: Always read the Read Me files and installation notes. It I see many problems reported that were clearly dealt with in the Read Me files, or installation guide. Besides, you never know when you'll get something out of it. IBM, for example, in their 45 - page readme for the OS/2 2.1.1 update, included on the next to last page, the instructions for finding, and installing Mah-Jongg. No where else was this information found. It was kind of a nice reward for diligence.)

You pick the drive to install onto, agree to the license agreement, and go. There's no options for the install, so custom installs don't apply, yet. On a freshly formatted partition on my PowerBook, the install took

John Welch <jwelch@aer.com> is the Mac and PC Administrator for AER Inc., a weather and atmospheric science company in Cambridge, Mass. He has over fifteen years of experience at making computers work. His specialties are figuring out ways to make the Mac do what nobody thinks it can, and showing that the Mac is the superior administrative platform.

about 15 minutes. Once the installer is done, the Setup Assistant fires up, and walks you through entering the base information to set up the Mac. It's pretty much the same as the Mac OS 9 Setup Assistant, except for entering an administrator username and password. This password is also used as the password for 'root' or the Unix super user ID

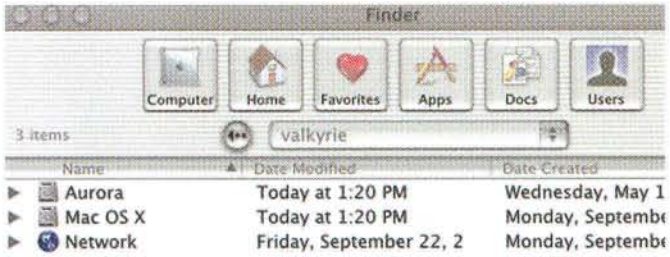
This brings us to another difference in OS X, the concept of the super user. The super user, or root, is the Unix equivalent of god on that machine. If you can log in as root, there is nothing you cannot do on a Unix machine. Literally, nothing. Do you want to rebuild the kernel? Root can. Do you want to delete every file in \etc, which is the directory that holds all of your configuration files? Root can. The point here is that root is a very powerful and very dangerous, so you want to be very careful about that user id. You should **never** log in as root unless you have a specific need, and then you should log out as soon as possible.

The Interface

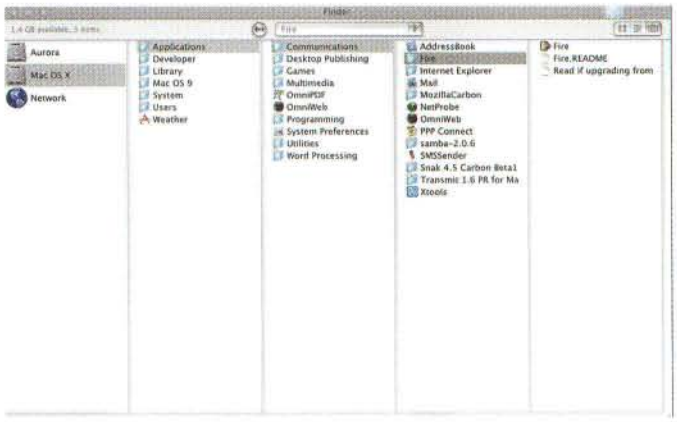
Once the setup assistant is done, Mac OS X reboots, and you are presented with the login screen. Once you log in, using the user ID and password you gave the setup assistant you are on the Mac OS X desktop, and ready to roll. One of the first eye-catchers, or at least mine, wasn't the Dock, but rather the lack of my hard drives on the desktop. There are a number of arguments for and against this, but for me, the two seconds it takes me to put an alias to the drive on the desktop renders it somewhat moot. Creating an alias requires the exploration of the new Finder.

The Finder

In Mac OS 9 the Desktop and Finder were interchangeable words; but in Mac OS X, the Finder is part of the Desktop application. The Finder, as in previous systems handles file-system functions. Visually, in the Finder window you see a number of buttons, similar to Sherlock 2, that are shortcuts to various places on your Mac OS X drive. These represent folders you would need to go regularly, such as Favorites, Applications, Documents, and a new one, Computer, as shown below. If you don't like the buttons, then ⌘(command)-B toggles them on and off. Also, if you have placed a folder in the Dock,⌘-clicking it opens it with the buttons hidden. Both views are shown below.

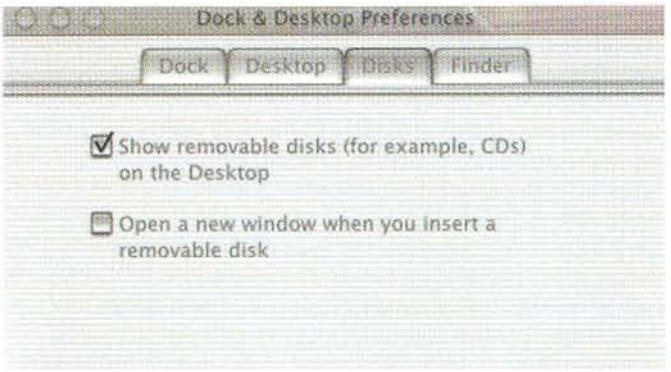


The Finder at Computer Level with the Toolbar showing



The Finder at Computer Level with the Toolbar Hidden

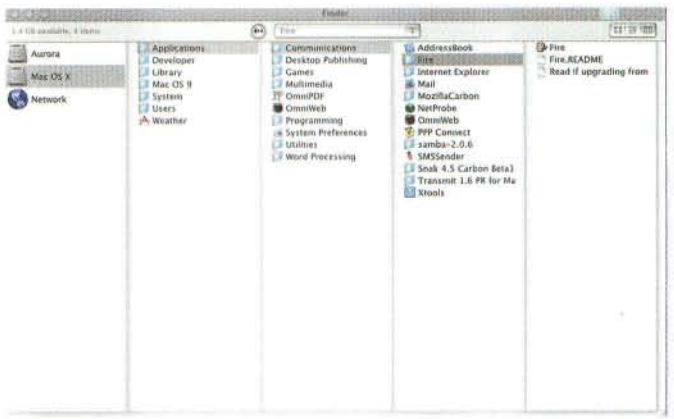
Computer is the root level of the system, and should be thought of as looking at the computer from bottom of the hierarchy, in that everything is above you. This level shows you all your disks, along with a new entry, Network, which we cover later. This view of the drives has caused some consternation, as previously, these items lived on your desktop. Well, the reason for it, although it doesn't apply in a standalone machine situation, is if you are accessing machines on a network, and you log into that machine, you are going to see a view that is a container for all the accessible shared drives. This network-centric view is essentially what Computer is giving you. The advantage to this view is that if you are in a heavy networking environment, you don't have to change view modes between your local Mac and Macs on the network. The disadvantage to this is if you are in a standalone situation, you could care less about the network view. Fortunately, in the Desktop and Dock preferences, you can set, your removable media to automatically show on the Desktop, as shown below.



Preference panel for displaying removable media on the Desktop

This is nicer, and less jarring to those of us not used to the NeXT way of looking at things. I think Apple would do well to add a "Show Internal Disks on Desktop" option here though.

Another of the changes in the Finder is the Browser view, shown here.

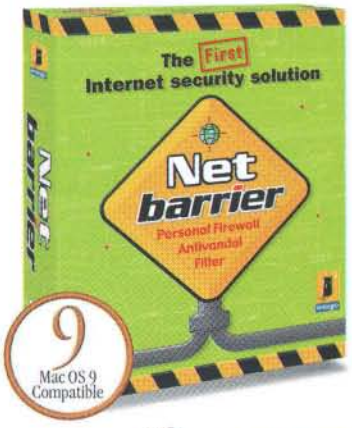


Browser view in the Finder showing path to the folder with the Fire application

The Browser view is a side-scrolling, multi-paned window that shows you your current location, on either the local hard drives, or the network. If you have a folder selected, it shows you all the items in that folder in the rightmost pane. If you have a document selected, it attempts to show you a preview of the document, along with the standard Get Info information. If you have an application selected, you get the generic Get Info details on the application. Although a little jarring at first, I have actually grown rather fond of the Browser view, it is fast, easy to use, and the ability to backtrack that easily through what can be many, many layers of folders is more than a little sweet. (To all the NeXTies — yes, you told me so.) I also find that some of the other modifications to

You think the Internet is safe. Think again...

NetBarrier. The first Internet security solution for Macintosh.



All Macs connected to the Internet (dialup, DSL, cable-modem) are exposed to hackers. Whether you are a home user or a professional user, your data interests them. That's why you need a security solution that only NetBarrier can provide.

Personal Firewall

NetBarrier protects and monitors all incoming and outgoing data. A customized mode allows you to create your own defense rules, thereby offering the most secure level of protection.

Antivandal

NetBarrier blocks all attempts to break into your Mac, detects wrong passwords and logs vandal attacks for complete protection. Moreover, it has an alarm to inform you of every intrusion attempt.

Internet filter

NetBarrier analyzes data as it leaves your computer and prevents unauthorized exporting of private information such as credit card numbers, passwords, sensitive data and more...

Available @ DeveloperDEPOT
<http://www.devdepot.com>
Toll Free: **877-DEPOT-NOW**
Outside U.S./Canada: **805/494-9797**

www.intego.com

intego

MacNN ★★★★

Macworld

Macworld 4/5

APPLE WIZARDS

the Finder windows, while jarring in some ways, make accessing the full features of the Finder windows more intuitive. The new drop-down list that shows your location in the folder hierarchy, has always been available via command-clicking the title bar, but this always struck me as too useful a feature to be hidden away as some power-user trick. Users need to be able to see where they are regardless of the current Finder view, so placing this feature out in the open is good. I also like the addition of a Back button, which makes this feature obvious without having to know oddball key combinations like ⌘U. I know that experienced Mac users are saying, "But that's so intuitive already." Well, for the true novice it isn't, and making it more obvious and intuitive does not distract from the power of the Desktop. It just makes it available to more people, and quicker.

Preferences Setting for how the Finder pops windows

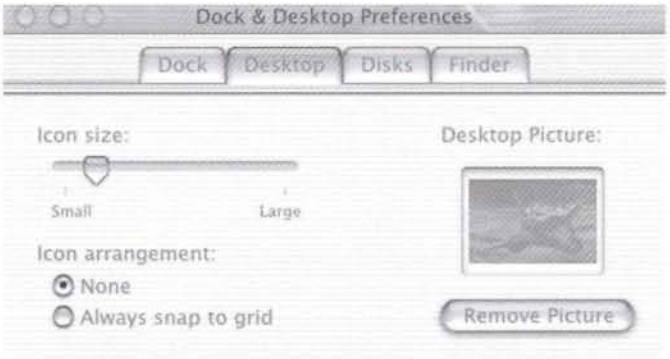
I also like the choice between having only one Finder window as you navigate, or the more traditional multiple windows. As someone who lives with their finger firmly planted on the Option Key, I'm very glad that Apple gave me this choice. There are, of course, some issues with the new Finder window. I would prefer the title bar to tell me the directory name, instead of the application name. I know I'm in the Finder, but what directory am I in? Keyboard navigation is inconsistent, for example, ⌘-Y is gone, so you have to revert to using ⌘-E for eject to dismount network drives or removable media. That's a little strange, as I'm not ejecting the network drive, (at least I better not be), I'm removing it from my machine for a while. I also would like the browser view to scroll as I drag items back from the current location, and better yet, forward to new locations, which would make up for the missing spring loaded folders. I find that most of these issues have the feel of beta bugs, more than eliminated features, so I'm not too worried.

The Desktop

The next set of changes is in the Mac OS X Desktop. At first glance there seem to be a lot of changes, but, in fact, I couldn't find that many actual changes. If you have been using OS 9's Multiple Users feature, Macintosh Manager, or NetBoot, much of the way OS X's desktop works will be familiar. If you have been using your Mac in a single user mode, then some of OS X's desktop will seem a bit odd. First of all, each user has their own Desktop Folder. This is so that you don't have situations where someone accidentally deletes or rearranges your desktop. They can't get to it unless you allow them access. This is annoying for the single user, , although when we look at the system preferences, the workaround will be clear. For the networked, business, or home user with the entire family using the same computer, this is a good feature, and is by far better implemented than in the current Mac OS's versions. This is not a surprise, as Unix has always supported multiple users this way, and OS X carries this UNIX-type support into the Mac OS. Also, in the current beta, you can't rename drives. Although there are pathname issues for this from the Unix perspective, this is a behavior that Mac users are used to being able to perform, and it should carry over into Mac OS X. If there is a reason for it not to be there, then it should be clearly articulated as soon as possible.

Another obvious change is the location of the Trash icon. It's no longer on the desktop, but rather on the Dock. This is a different location, but the functionality is still the same. You drag files to it, open it up to remove things, unmount media by dragging it to the trash. ⌘-backspace places things in the Trash, and shift-⌘-backspace empties the Trash. By placing the Trash on the Dock, it can't get hidden or lost behind other windows, which is good for new users. Considering the time I've wasted over the years working with cluttered desktops, and typing t-r-a as fast as I could, it's a good move for experienced users as well.

The icon size on the desktop is independent of the screen resolution and **that** is a feature that is sure to be greeted with joy by the vision impaired everywhere, including yours truly. I like the fact that I can have room for the way I work, and still be able to resize my icons to where I can see them without having to drop the resolution.



Desktop Preferences showing Icon Size settings and Desktop Picture Settings

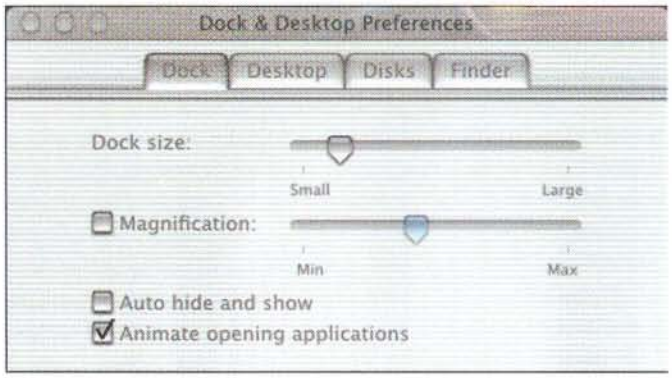
There are some interface issues with the Desktop that seem more like beta bugs than anything else. Auto sorting by name is not enabled, and the Desktop doesn't refresh its contents as fast as it should, especially if you use the command line to add things to the Desktop. You cannot have slashes in the names of files, which is a limitation of OS X, not HFS+. I understand that Unix uses slashes as directory delimiters, but this is not Unix, it's based on Unix, and this is a restriction that may cause problems. (Note: There were also applications in previous Mac OS versions that had trouble finding files when slashes were used in filenames.) On the other hand, you also now get 255-character filenames, so the news isn't all bad. Finally, the system font and font size cannot be changed. I understand that the font size can be changed by scaling the icon size, but some of us want tiny icons and big fonts, or vice-versa. We also want a font that isn't Lucida Grande. These are annoying problems, but fixable, and I imagine they will be fixed in the release version of Mac OS X.

The Dock

On to the most controversial part of the new OS, the Dock. This is an amazingly polarizing application, and yet, like much of OS X, it really does grow on you. I think much of the problem is what it's replacing: the Apple Menu, the Applications Menu, the Menu Bar clock. The Dock also contributes to the loss of tabbed folders, which for some is more traumatic than others. Now, lets take a look at the Dock, or at least my dock after login

The Dock is a bit smaller here than on my normal screen, but, even on my PowerBook's display, there is a lot of feedback here that you don't get by default in Mac OS 9. First, I know exactly what applications are running, in this case, from left to right: Finder, Stickies, Console, Classic Menu, wClock, and Grab. Classic Menu is an Apple Menu replacement for OS X, and wClock gives me a menu bar clock, and calendar as well. (Note, the Public Beta represents one of the biggest shareware and utility goldmines for Mac developers, and I am pleased to say that there is a wealth of items out there already.) I also have a number of applications handy that I use frequently. The four items on the left are the Trash, the What's New Help file, the link to the OS X feedback page at Apple, and a link to my Applications folder. In addition, although I couldn't get screen grabs of this, ⌘-Tab cycles through the open applications, and pops

their names up so I can see which one is selected. Similar to the Desktop icon size, the Dock icon size can be set independently of screen resolutions, as shown below.



Dock Preferences Panel

I have the magnification, and hide and show features turned off, and the opening animation turned on. Now one thing that you may notice that is not in my Dock is the clock. I tried storing it in the Dock, but it was too small. I tried the floating option, but it was in the way all the time, plus the icon running in the Dock, took up space in two places. So I now use wClock, which gives me a menu bar clock, and in conjunction with Classic Menu, makes my menu bar look quite familiar. The clock on the menu bar is very handy, and an option to have it there, floating, or in the Dock would be appreciated by many users.

Menu Bar with wClock and Classic Menu running.

There is another feature to the Dock which has only been hinted at, but in fact is a major plus , and that is the ability of the Dock icons to display live data. Aside from the endless demonstrations of QuickTime movies in the Dock, there are some other nice applications that make good use of it.

Mail Application Dock Icon showing unread mail symbol

The Mail application uses the Dock icon to indicate you have unread mail in your inbox. Very nice, and handy if you don't want to have your Mail window always open or maximized. Another application that makes use of this ability is the CPU Meter application. If you minimize both windows, the Dock icons update the usage graphs in real time, giving you a miniature, yet live display of how hard you are hitting your system.

Dock showing live CPU Meter icons

So while the Dock may not be the end all application , it is a lot more capable than people give it credit for. I hope that application developers take advantage of its abilities.

Setting Up OS X

Enough of the basic user interface, let's take a look at how you set your system up. Almost all system settings are accessed through the System Preferences application. After about a week of exploring Mac OS X, I found out that Apple has done some pretty neat things. First off, the System Preferences are doing a lot of the low level Unix configuration file editing. For example, as you go through and set your network preferences in the Network control panel, you also alter the NetInfo settings. NetInfo is the way that NeXT networks kept track of machines, users, accounts, printers, and access rights, plus everything else. It is analogous to Novell's Directory, or LDAP (Lightweight Directory Access Protocol), and is administered via the NetInfo manager application. However, as we will see later, this is not a very intuitive application, and setting the wrong thing can hurt your configuration, and prevent your Mac from booting. NetInfo also holds information for files like hostconfig, which is where the BSD Unix layer gets its information for things like host names, and IP addresses. From what I can see, System Preferences modifies NetInfo, which modifies the configuration files. This may not be totally correct, but not having to use a terminal and EMACS, or worse yet, vi, to set up your Unix settings; indeed, never having to even know where hostconfig lives, is a sign that Apple has put a lot of work into taking the Unix fears out of Mac OS X.

System Preferences

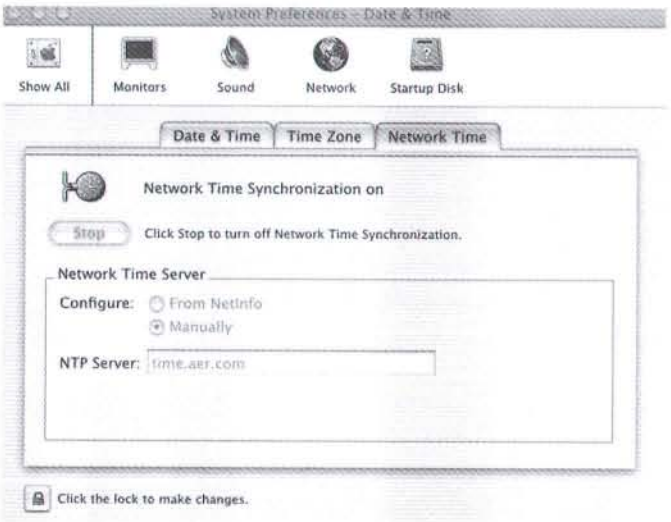
Now let's take a look at a few of the features in System Preferences. The application reminds me of the Mac System 6 control panel, where one application held all your system settings. Although the OS X System Preferences application is a nicer version, it is still a single, coherent place for system, not application-specific settings. The full view of the app is shown next.


development revolution.
You're part of the movement.
At the core you'll find two innovative technologies allowing you to deliver the most comprehensive software solutions available – 4D and WebSTAR.
Time is short. Seize the moment.
Maximize your efforts using revolutionary development tools and Internet Servers that deliver now and into the future.
Don't waste today.
Download your future. Ignite your potential.
4D
WEBSTAR.4D.COM
1.800.881.3466
© 2000 4D, Inc. All Rights Reserved. 4D and WebSTAR are registered trademarks of 4D, Inc.

'Show All' view of System Preferences

For the most part, these look familiar, so we won't take a look at all of them, but I will cover the settings of interest to network administrators.

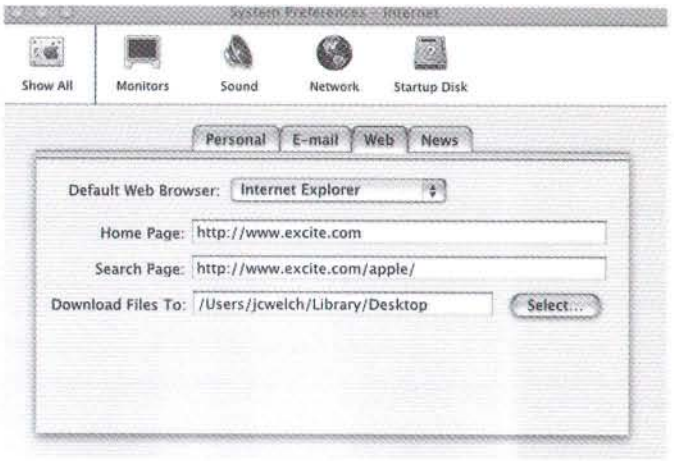
The first one is the Date & Time control panel:



Date & Time Control Panel Network Time tab

The first two tabs, Date & Time and Time Zone, have the same functionality as the Mac OS 9 versions. The Network Time tab is interesting though. It gives you the ability to either manually enter a network time server's domain name or IP address, or you can select From NetInfo, and let NetInfo handle where on your network your Mac gets its time information from. This feature is a boon to anyone trying to change this setting on a network with a few hundred Macs.I hope that Apple allows for NetInfo to integrate with other directory services too, so that this type of administrative tool is not limited to NetInfo. While an excellent way to manage a network, NetInfo is hardly the one of the most common tools.

The next control panel of interest to network administrators is the Internet settings panel. This illustrates Apple's attempt to bring the functionality of Internet Config into Mac OS X.



Web settings of the Internet control panel

The other reason I am highlighting the Internet preferences panel is to show that:

1. In the Web tab you have the same flexibility in selecting your default web browser in Mac OS X as you do in OS 9.

2. You can select where to Download Files To, so that with the multi-user capabilities of OS X, I, and anyone else who uses my PowerBook under OS X can have our web browser download to the desktop, yet not interfere with anyone else's desktop.

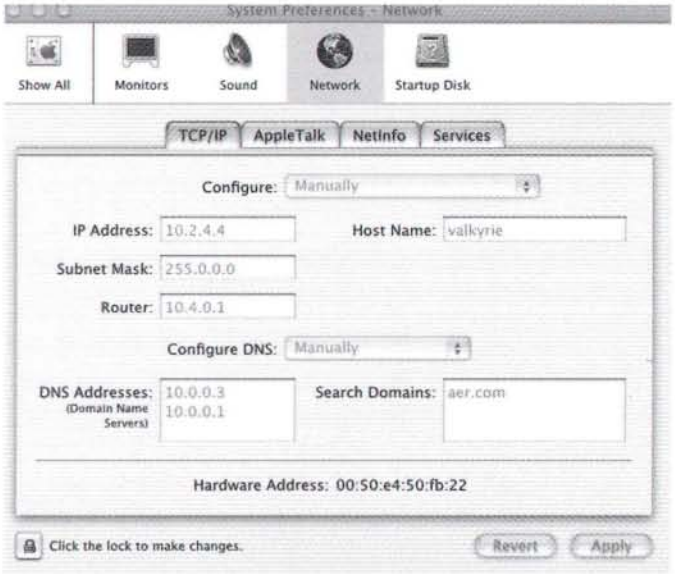
This is nothing completely new, but the fact that it is an integrated feature of the OS, instead of a bolted-on kludge will result in a smoother user experience. Do expect changes here though, as Mac OS X uses the word "E-mail" even though their own style guide uses "email."

Login control panel showing the Login Window Settings

Login control panel showing Login Items

The next panel of interest is the Login preferences panel. This controls Login functionality, such as automatic log in in the Login Window tab. Plus you can choose your enabled startup items in the Login Items tab.

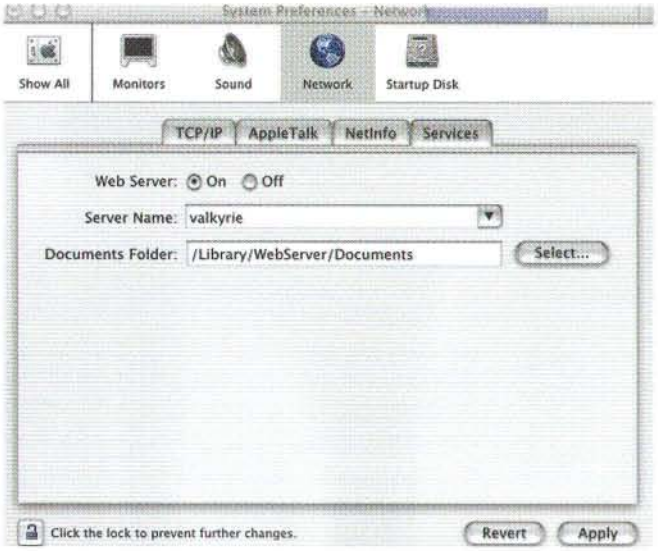
The reason for moving the startup items is based on the multi-user capability of Mac OS X. If you have twenty people sharing the computer, and they all have 3 different startup items, you wouldn't all the applications to start for everyone. tThe applications only start up when you log in, and that way only your choices start up. This is also the location of another beta bug. Once you set up the link to an application, if you move the application, the link is broken, indicating that Unix-style hard links are being used, instead of proper aliases. This can be quite annoying, but I doubt that Apple would leave such an interface bug like that in the final version. Also worth noting is that if you are the only user for a given machine, you can set it to automatically log in for you, so that you do not have to go through the log in process. Another option in the Login Window tab disables the Restart and Shut Down buttons. This is useful if aMacintosh computer is being used as a server, the only way to restart it would be through the hard switches on the computer, or by logging in as root. One note here, if you do have the automatic log in enabled, you should **never** set it to root. As the super user for a given system or network, there is **nothing** that root cannot do, and therefore that user should **never** be the default user for any machine.



TCP/IP tab of the Network control panel

The third panel we will look at is the Network preferences panel. TCP/IP is the first tab, and looks familiar with the exception of the Host Name. The host name is used to identify the machine, not only via DNS, (in my case, valkyrie.aer.com), but for AppleTalk and NetInfo as well. What isn't shown, and unfortunately I couldn't get a good screen shot is the fact that the TCP/IP panel has configuration settings. I have had two separate static IP addresses on this PowerBook, and if you could see the Configure: pop-up menu, you would see: Manually 10.2.4.4, Manually 10.2.4.1, along with DHCP and BootP. This configuration support is a good sign that Location Manager is not gone, as many fear, just not completed yet.

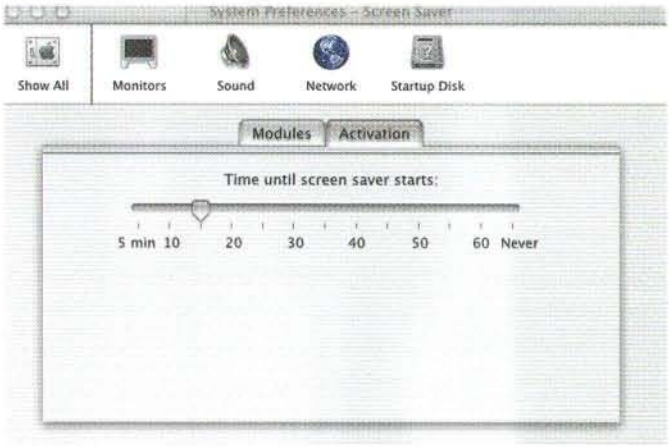
The next tab is AppleTalk, and is fairly simple. AppleTalk is toggled on and off with a checkbox and you type in the AppleTalk Computer Name. The NetInfo tab simply asks if you want to connect to an existing NetInfo domain or not. The fourth tab is the Services tab, and that bears a little more investigation.



Services tab of the Network control panel

The Services tab handles the Web Server that is included with Mac OS X, namely Apache. Well over 60% of the Internet is run off of Apache, and with the full version of that web server, Apple is giving Public Beta users a very powerful way to publish documents and files. This also makes the initial decision as to which web server to use with Mac OS X very easy, but it creates an interesting business proposition for the WebSTAR folks. 4D, Inc. now needs to give their existing users a compelling reason to stay with WebSTAR, and create a more compelling reason for people to buy WebSTAR instead of staying with a free product that is every bit as powerful. However, the Services tab only turns Apache on, and lets you decide where you want the documents to reside. It doesn't configure Apache, or make it secure, so there is still a lot of work left for the user to do. This also is where Apple, or third party developers can come in, and create products that help you configure Apache in a way that makes the Mac community comfortable with such a powerful tool. Another advantage to including Apache is that it finally gives the Macintosh Web community access to all the powerful Apache tools , including, potentially the Apache add- ons that allow you to use Apache as a server for Active Server Pages, instead of Microsoft's Internet Information Server, IIS. Opinions of Microsoft aside, IIS is a major player in business intranets, and Apple has always been forced out of this large, lucrative arena. Apache gives them a toehold here, and a fairly respectable toehold at that. Considering the amount of Web content created on Macs for other server platforms, it's about time that that content can be served on a Mac without the potshots directed at the Mac OS.

The next stop in System Preferences is the Screen Saver panel. Although not thought of as an administrator kind of application, the fact is, most servers have a timed screen lockout system. The settings for the screen saver do not currently allow for forcing a password to get past the screen saver, which makes it less useful to administrators. Apple, or a third party developer should be able to provide a solution.



Screen Saver control panel Activation tab

The Sharing preferences panel is next on our tour. This controls a number of items of interest to users and administrators.

Sharing control Panel

The File Sharing button controls Mac OS X's Apple File Protocol (AFP) over TCP/IP sharing capability. The Public Beta only allows you to share your Public folder, but like a lot of this version, this looks like a beta bug, and should be fixed quickly enough. The other two checkboxes are new to Mac users. The first, Turn on remote Telnet access, enables the Telnet server that ships with Mac OS X. This is a very handy feature for administrators, as it gives them the ability to log into a Mac OS X machine, and perform any of the


• Custom Destinations
• Electronic Transaction Processing
• Easy Trialware Creation
• Install or UnInstall
Get It Together With InstallerMaker™
• Installation From FTP or HTTP Sites
When Time Is Money, Get It Done Fast!
Build Smaller Installers – With StuffIt InstallerMaker, you'll build installers faster than ever before! Compress your installers an average of 15% smaller using the power of the StuffIt Engine®. Smaller installers download faster, provide added space on CDs and servers, and increase network bandwidth.
Quickly Create Demoware - Easily turn your application into a polished demo. Just set the number of days for the demo to be active, paste in the graphics, and you're done!
Archive Freshening - Automatically update your installer project file; eliminate repeated searches for modified files.
• Resource Installation
• Built-In Resource Compression
Scripting Saves Time - InstallerMaker provides full scriptability for all features. Automating the build process saves time and helps ensure the integrity of your installer.
• ShrinkWrap Disk Image Support
• Marketing Opportunities To Help You Make Money
Trialware in Minutes - Creating trialware is a breeze with InstallerMaker. With just a few clicks, and no extra coding, you can create trialware that is e-commerce ready.
Update in a Flash - Easily build intelligent "diff" files in installers to create small updaters for quick online distribution. From one simple installer, you can update 68k, PowerPC or FAT applications.
• Hierarchical Package Support
More Details Online - There's a lot more InstallerMaker can do for you. Get all the info at www.aladdinsys.com.
• Built-In Updaters
Aladdin Systems
© 2000 Aladdin Systems, Inc. StuffIt InstallerMaker is a trademark of Aladdin Systems, Inc. Other products are trademarks of their respective holders.

standard Unix remote administration tasks that can be run from a command line. This is also a dangerous feature for the very same reason. If a computer cracker were to Telnet to an OS X box running Telnet services, and be able to log in as root, again, this person would have complete control of the Mac OS X box from a command line, and then do essentially, anything that is possible from a command line, which in Unix is quite a lot. (I don't want to sound too alarmist here, as having Telnet access to OS X is an incredibly useful feature, especially to those of us who are network administrators. As with any feature, it has its downside as well. The command line, and command line access are not inherently bad or good, but like any tool, can be used both ways. Mac OS X ships with Telnet turned off, and unless you have an explicit need for this feature, I would recommend keeping it off. A lot of the security of the Mac OS has been due to the lack of remote access features in the OS. While this has saved a lot of Mac administrators from having to deal with crackers, it has also restricted administrators from the kind of capabilities that our Unix and Windows compatriots have. Services, daemons, and applications like Telnet, and other Unix capabilities are going to give us a lot more power, and a lot more things to worry about. It's better to start now, and be ready, than to find out the hard way when some script kiddie turns your machine into a DOS attack zombie.

The other setting in the Sharing settings panel enables the FTP service. So your Mac can be a FTP server out of the box, no extra software required. Again, my warning about security applies here as well, if you have either of these services turned on, be sure to pick a good password for root, and change it often. Even with Telnet turned off, a cracker with root access could FTP to your machine, and replace the configuration file that turns off Telnet with one that turned it on, and a script that would notify him or her that access was now on. The next time that machine rebooted, that cracker would have full access to your Macintosh. Ssome common sense keeps you safe, if you don't need to be an FTP server, leave that service turned off. If crackers can't get in to your machine, they can't make use of it. If you need this service turned on, then change your root password regularly, and make it incredibly cryptic. This is one area where an ounce of prevention is worth a ton of cure. The final section of the System Preferences we will look at is the Startup Disk preferences panel.

Startup Disk control panel

The Startup Disk preferences panel is where you choose the operating system you wish to boot from, whether that be on a separate disk, as shown above, or on the same disk. Like other settings in the System Preferences, you have to be an administrator on the machine to change any setting in this preferences panel.

Other Settings

There is one other place that is normally used to set up Mac OS X, and that is Multiple Users. (I am leaving out the Keychain setup, as it seems to be unchanged from Mac OS 9.) This is where you would normally create, edit, and delete user accounts on your Mac OS X machine. Multiple Users is a much simpler beast than its OS 9 counterpart, and is more intuitive to use.

Multiple Users main screen

One of the first things that you notice is that one account is missing, namely root. This is a good thing, because it means, that even if I unlock Multiple Users to make changes, I cannot touch root, either to delete it, or change its password. There is an interface for altering root, which we will look at in a bit. Multiple Users allows you to add, edit, or delete users. You must be an administrator to gain access to its features. Once inside the application, editing a user is fairly

simple. As shown below, you can change the Name, Short Name, (the userid), the Password for the user, and decide if that user can be an administrator . If the user is an administrator, then their userid and password will unlock those settings panels and applications that can only be used by an administrator. Granting this privilege should not be done unless that person needs that kind of complete access.

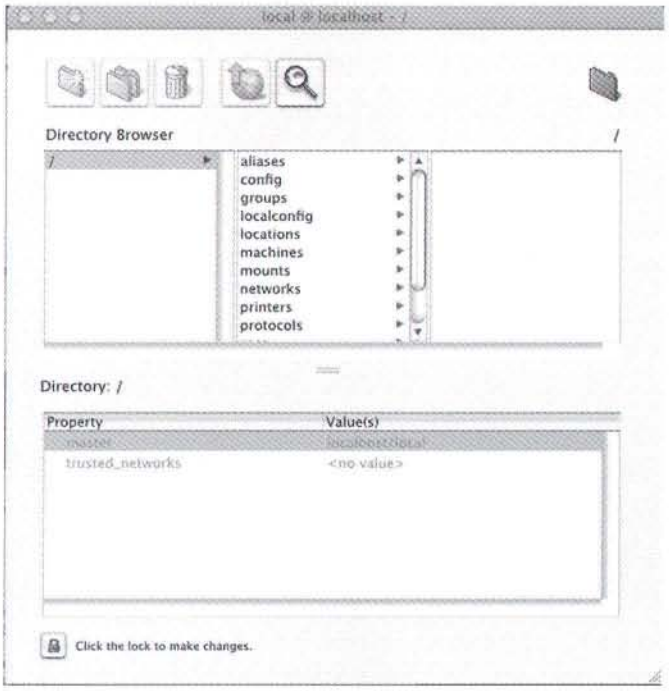
Multiple Users showing the options for editing an existing user

The screen for creating a new user is identical to the edit user screen, except that when you open it, none of the fields are filled in. This discussion takes care of the common settings interfaces that you will use for Mac OS X. Our next step is to take a look at the less common interfaces for setting up Mac OS X, namely NetInfo and the Unix settings files.

NetInfo

NetInfo is the administrators tool for setting up not only individual OS X, OS X Server, and NeXT machines, but for configuring networks as well. It is an amazingly powerful tool, and used correctly, can make much of an administrator's job easier. Unfortunately, outside of the NeXT and Mac OS X (Server) community, NetInfo is hardly used at all, being superceded by products like Novell Directory Services, NIS+ from Sun, Active Directory from Microsoft, and LDAP. This is not to say that NetInfo isn't a capable tool, quite the contrary–it can easily match and in many cases beat the capabilities of these other systems. Since NetInfo is a minority player, Apple needs to do much more work on integrating NetInfo with these other systems, particularly NIS+ and LDAP, as these are the major players in the Unix world.

The application that administers NetInfo is the NetInfo Manager. When you first open it, you are presented with the root, or / view of localhost, which is Unix for the machine you are sitting at. If you have the proper privileges, you can also use NetInfo Manager to manage entire NetInfo network domains, but for this article, we'll stay on the local machine level.



Initial NetInfo Manager Screen

As you can see above, the NetInfo Manager uses the same type of browser interface as the Finder. The / indicates the root level of the computer, analogous to the Computer level of the Finder. The middle pane shows the items that are controlled by the NetInfo manager, and although we won't cover all of them, some of them are worth discussing. First we should look at some of the items in Config, such as AppleFileServer and ntp. If we look at the AppleFileServer screen below, we can see that NetInfo does give us access to the settings we need, even if the interface leaves something to be desired.

Join your friends
and colleagues in the Mac community
at The Largest IT event
on the West Coast!

World-Class Exposition!

Macworld Conference & Expo/San Francisco 2001 is a One-Stop-Shop offering discounts not found anywhere else!

- **Featuring more than 500 exhibiting companies**
- **Thousands of new products, software and services**
- **Hands-on demonstrations**
- **Test-drive new technologies**
- **Awards and competitions**
- **Prizes, drawings, giveaways**

Special Interest Boulevards

Macworld Conference & Expo offers so much for every Mac user!

Check out the Special Interest Boulevards while you are exploring the exhibit floor to find and compare hot new products for your particular needs. Popular areas include:

- **Digital Multimedia**
- **Music and Audio**
- **Developer Central**
- **Net Innovators**

Be sure to visit www.macworldexpo.com for exciting additions!

Cutting-Edge Educational Programs!

Conference sessions for the New, Beginning, Intermediate and Advanced users!

Macworld/Pro features technically in-depth presentations and issues-oriented discussions about professional applications of Macintosh technology. The Pro conference offers ***the most sophisticated training available*** on Mac networking, digital video, art director/creative management practices, and Mac systems administrations for large organizations.

Macworld/Users continues to be one of ***the best educational values anywhere.*** The Users conference features industry experts offering skill development on the most popular Mac-related tools and professional development courses for users who rely on Apple technology to gain a competitive advantage.

MacBeginnings **– San Francisco debut!** After great success this July in New York, MacBeginnings will make it's debut in San Francisco! These high-energy, informative sessions will provide first-time attendees, new and beginning Mac users a starting point to enter the Mac community. MacBeginnings are free and open to ***ALL*** registered Macworld Conference & Expo attendees!

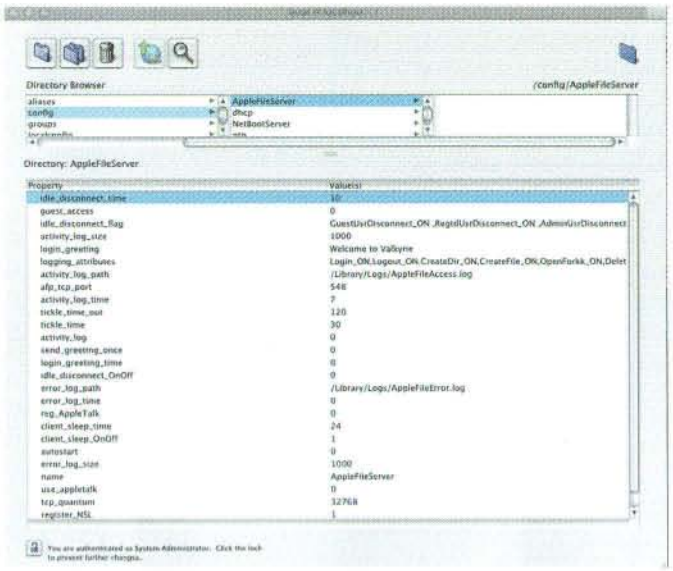
Macworld
Conference & Expo
www.macworldexpo.com

Owned and Managed by
IDG WORLD EXPO

Flagship Sponsors

Macworld MACBUYCOM Macworld.com

MacWEEK.com MacCentral.com

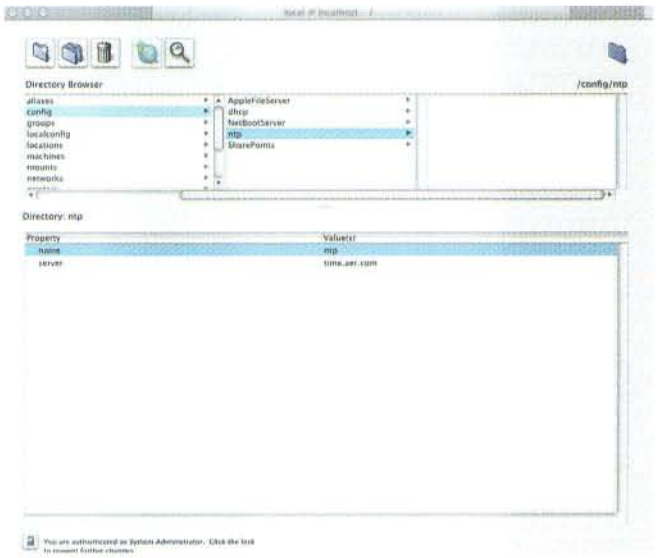


NetInfo AppleFileServer Settings

To me, and admittedly I am not a NetInfo expert, or even a NetInfo power user, this is a mix of good and bad interface components. Good, because I have quick access to any setting that an AppleShareIP Server needs to have, and it's in a consistent GUI. Bad, because since there is no NetInfo Manager help in the Public Beta, I am forced to guess, at what are some of the settings. It may be obvious that the idle_disconnect_time should be talking about minutes, but it would be handy to have that made more obvious. Nonetheless, the NetInfo Manager does give me the features I need to set up this service on my Mac OS X machine. The guest_access setting is less obvious, but one would suppose that this setting uses a binary setting, since other settings in this pane dol, so for now, guest_access is turned off. Moving down, we can see that the idle disconnect settings for all users, regardless of level, is turned on as well. The activity_log_size is set to what I would guess to be around a megabyte in size, as if it were a kilobyte, then it would not be able to hold many entries. We can also see the login greeting for this machine, and what activities are logged. It is good to note that almost all activities are logged. If you did suspect someone has cracked your machine, good usage logs are some of the best ways to find out what is going on, and who is doing it.

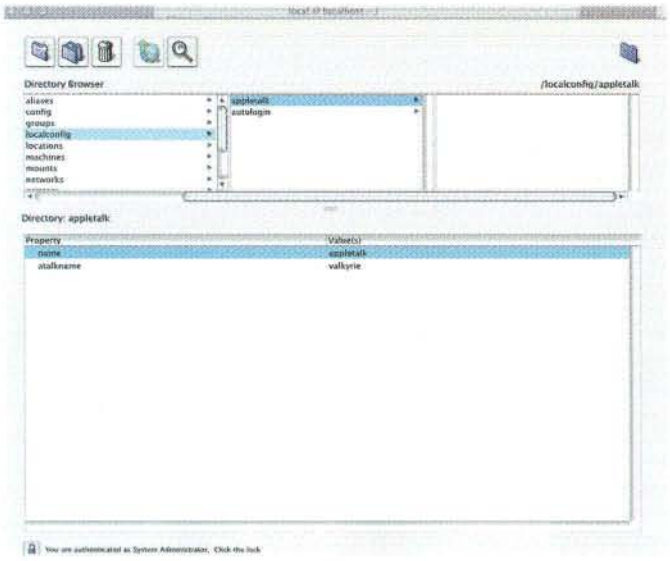
The path where the activity log is kept is the next setting, and although mine is left at the default location, the wise administrator will change the default so that a cracker cannot alter the log to hide their tracks. The port used for AFP over TCP/IP is the next setting, which is needed if you want to set up a firewall to allow this protocol through the firewall. Moving down the list, the idle_disconnect appears to be turned off, a security weakness, were this an actual server. The reg_AppleTalk setting would appear to be the one to allow for straight AppleTalk to be used. As I have not yet confirmed this, I am leaving it alone. (This brings me to my next warning. If you don't know what a setting does in NetInfo, leave it alone. The NetInfo Manager is how you, as an administrator, can set up features on an Mac OS X machine that are not available in the conventional settings panels. It is also an easy way to send your Mac OS X machine into limbo at light speed. Any changes I have made in NetInfo Manager have only been made after I was sure that I knew what I was doing.) The final setting on this page that we look at is the register_NSL setting, which when turned on, as it is here, allows this machine to show up as an AppleShareIP server in the Network Browser application.

The other setting in config that we want to look at is the ntp setting. There isn't much here, but it does show you the relationship between the System Preferences and NetInfo. If you look back at the screenshot of the network time tab in the Date & Time control panel, you can see where I had manually entered a timeserver name. Looking at /Config/ntp setting below, we can see that the Date & Time control panel setting was automatically entered into NetInfo for us. Hopefully, Apple will continue to provide more intuitive interfaces for other NetInfo settings, limiting the amount of time non-administrators would need to spend here.



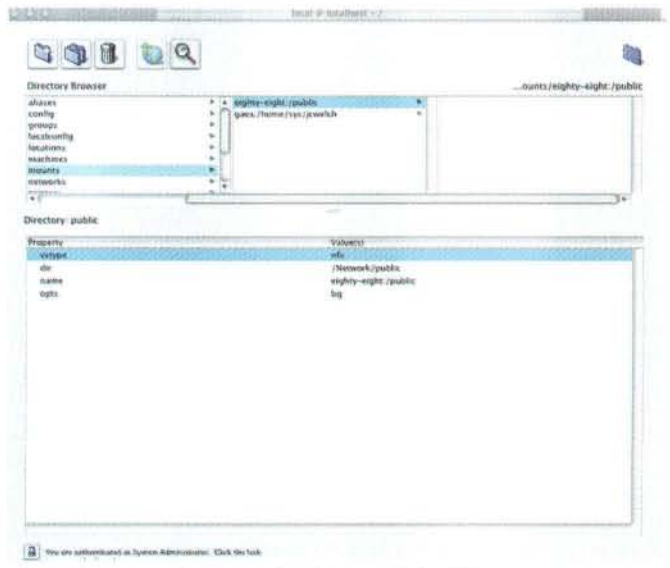
NetInfo settings for /config/ntp

Another example of how the System Preferences set NetInfo parameters is the /localconfig/appletalk setting. This is what is set when you enter a machine name in the AppleTalk tab in the Network settings panel, as you can see next.



/localconfig/appletalk settings

It's neat to see how Apple can make dealing with NetInfo more simple. What about things that aren't in the System Preferences, such as Network File System, NFS access? Since Mac OS X is based on Unix, and NFS is the standard way for Unix systems to share drives and directories across a network, how do you do that? Well, this is one place where NetInfo saves you from setting up NFS automounts by directly editing config files. As we can see in the example below, I have two NFS shares set to automatically mount when I log in to Mac OS X on my PowerBook.



/mounts/eighty-eight:/public

The first parameter of each entry is the vstype, in this case NFS. This tells NetInfo what type of drive it's going to be mounting, as there are different ways of handling different types of mounts. The next parameter is dir, which is the local directory where the mounted drive should appear. This is where Mac OS X's Unix base shows up. In Unix, when you are going to mount a remote drive via NFS, you not only need to know the computer, and the partition you wish to mount, but you need to tell your computer where this remote mount should be attached to locally. In most cased, with OS X, you want this to be in a subdirectory of Network, which we saw in the earlier Finder screen shot. Now within Network, there is a folder called Servers, but only the system can alter that directory, so I have set all my mounts to be subdirectories of Network. In the example shown, I call it public, so the value for dir in this case is /Network/public. Note that creating it here does not create the actual directory. You still have to do that either via the Finder, or the command line. Although experienced Unix administrators will have no trouble with this, Mac administrators, may find this inconsistency will trip them up. Hopefully, a more coherent interface for managing NFS mounts in Mac OS X will show up as the final release date nears. The next parameter is the name of the remote computer and the path to the directory that I wish to mount. This follows the Unix convention of <machinename:/path>. In this example, the machine name is eighty-eight, and the directory I wish to mount is at the root of the system, and is named public, so the value for name is 'eighty-eight:/public'. The final value is opts, for options, and the only thing entered here is 'bg', which tells the system to mount this share in the background. Any other mounts are set up the same way, and so far, every time I log in, the mounts have been there.

This brings me to another inconsistency in Mac OS X, the way it handles different types of mounted drives. If I connect to an AppleShareIP server, the share shows up on the desktop, and goes away when I disconnect. Any NFS shares created in NetInfo Manager seem to be always mounted, even if the computer is not on the network, because a permanent directory is created so that the NFS share has something to attach to. If you are used to Unix, this is nothing unusual. However, in the Mac world, if you aren't attached to that share, it has no business existing at all on your machine. I actually like this better, as it is a clear way of notifying the user that this or that share is available for use. Hopefully, this will be fixed, either by Apple, or an enterprising third-party developer.

The next part of NetInfo that bears looking at is the /groups section. These are Unix user groups, similar to the groups you would set up in any network. The ones shown below are the standard groups that ship with Mac OS X. Of particular interest to the network administrator is the wheel group This is the group you are assigned if you are created as a user that can administer this machine, sys, which on a stock install has root as its only member, and admin, which also shows root as its only member. Groups are an important tool in Unix security, and can be a great help to an administrator if set up correctly. Conversely, they can also be a nightmare to an administrator if used incorrectly. As you can see, there are groups that are application/purpose-specific,

"Never forget that the most powerful force on earth is love."

–Nelson A. Rockefeller

Move over love.

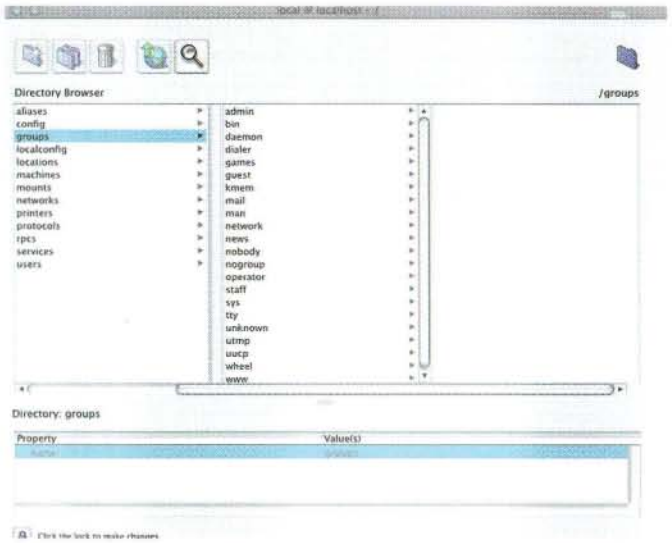
Get a new Apple PowerMac™ G4 or one of our refurbished models at price you will L-O-V-E.

Small Dog Electronics

1673 Main Street
Waitsfield, VT 05673 USA
Phone: **802-496-7171**
Online: **smalldog.com**

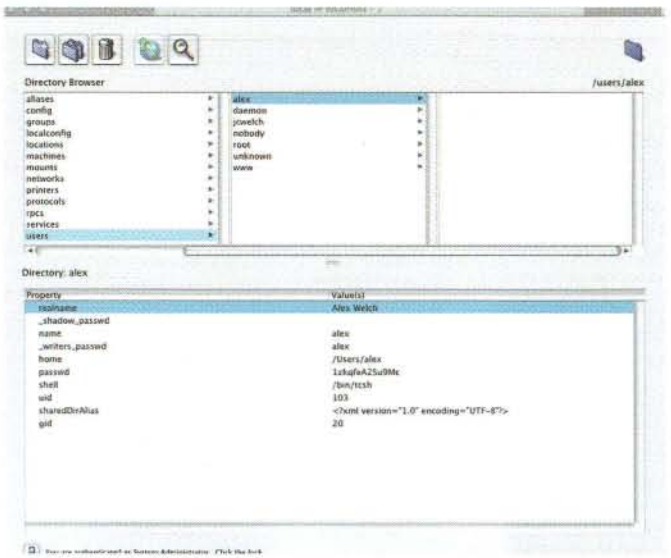
Apple Specialist

such as www, mail, news, etc. This is so certain types of applications, (and with Unix, the word 'application' is used very loosely), can have the low-level access they need to function, without needing to operate as root. Remember, root should not be used lightly or often. (If it seems that I am really beating this with a stick, I am. I have seen too many networks standing on their heads with all kinds of arcane passwords for users, limiting login times, disk quotas, etc, only to be easily hacked because the administrators were constantly logging into the system as root, remote logins as root were allowed, and too many applications were running as root. The fact that Apple disables remote root login by default in Mac OS X is a good sign that they want it to have the same level of out of the box security as OS 9 does.)



NetInfo /groups pane

The next part of NetInfo we look at is the /users section. This is where every user that is entered into multiple users shows up. The entry we are looking at is one I created for my son, Alex. It shows the information for his full name, and his login name. The home entry shows his home directory, /home/alex. The entry for _shadow_passwd is blank. From the information I could get on NetInfo off of the Internet, this is because NetInfo doesn't use shadow passwords, as they are less secure than NetInfo's method of storing passwords. The shell entry shows his default command line shell, should he choose to use it. It also shows his userid, 103, and his gid, or group ID, 20. Just like any other Unix, you can belong to more than one group, and the first number shown in the gid entry is the user's main group.



NetInfo /users/alex pane

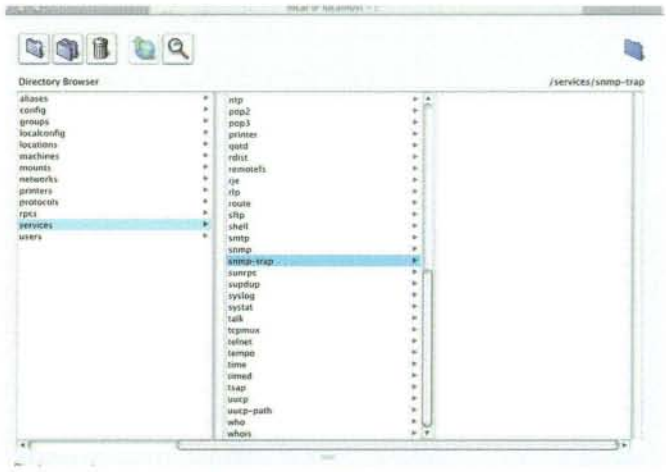
The sharedDirAlias is interesting, as it highlights the heavy use of XML in Mac OS X, along with how powerful XML can be. Some of the hacks starting to circulate that allow AirPort cards to function, for example, are nothing more than XML entries. Although it's hard to tell at this stage how much use of XML Apple is going to have in the final release, it's fairly extensive in the Public Beta. As an example, almost all your preference settings for Mac OS X are kept as .plist files, which are nothing more than XML files.

An example is the ApplePowerManagement.plist file, which is the where the Energy Saver control panel settings are stored:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist SYSTEM
file://localhost/System/Library/DTDs/PropertyList.dtd">
<plist version="0.9">
<dict>
  <key>DisplaySleepTimeIsIndependent</key>
  <true/>
  <key>HardDiskSleepTimeIsIndependent</key>
  <true/>
  <key>MinutesUntilDisplaySleeps</key>
  <integer>0</integer>
  <key>MinutesUntilHardDiskSleeps</key>
  <integer>30</integer>
  <key>MinutesUntilSystemSleeps</key>
  <integer>0</integer>
</dict>
</plist>
```

This is a standard XML 1.0 file, with a custom DOCTYPE, referencing a DTD that sets up the parameters for a property list. The entries that follow are fairly easy to read as well. It shows that display and hard drive sleep times are independently set. The Display sleep is set to zero minutes, along with the system sleep time, and the hard drive time is set to 30 minutes. This is a case of a beta bug as well, since the entries for the hard drive and system sleep times are nicely reversed. I'm not too upset, because the power management issues are pointed out in the Read Me.

The final part of NetInfo Manager we should look at is the services. We will not alter anything here, but it is good for showing that Mac OS X does ship with the standard Unix services, or ability to run them, even if all are not enabled in the Public Beta. Available services like pop3, rje, smtp, syslog, talk, telnet, timed, uucp, snmp, and whois, point to Mac OS X being able to be a full-featured player in the Unix and networking worlds.



NetInfo /services/snmp-trap

So How Does It Work?

Well, so far, quite well. The network speed is very fast, and a nice improvement over OS 9. As a test, I was able to achieve saturation speeds on a 100Mb Ethernet network for a 250MB file copy, and while that was going on, I was doing other transfers as well. The only slowdowns were due to network bandwidth limitations, not OS limitations. Web browsing, and FTP feel snappier than in OS 9. The ability to telnet into my Mac, and kill a process that was eating CPU was something to warm my administrator's heart. Even nicer was the OS telling me a process had been killed, but that everything was still okay.

From the Unix side of things, all the command line capabilities are there, and with the beta of XTools from Tenon sitting on my OS X box, and running Netscape on a Solaris box, and IDL on an SGI was just too sweet. It's a full Unix environment, and it works like one, and acts like one, both good and bad. Apple has just managed to make it better. All the low level Unix bits are there, including, /etc, /var, sendmail, and hostconfig, but you never have to see them if you don't want. In the two weeks of constant use, I've only been in the config files by choice, not by necessity. The System Preferences and NetInfo take care of that for you. In fact, thanks to NetInfo, many of the Unix config files aren't even used, but are there as a backup system. AppleScript is even there, although it is very rough as of yet. I plan to look heavily at that very soon.

The fact that Mac OS X gives you all of its power and features in a uniform, coherent interface is the most important part of the operating system. The Unix underpinnings, which are powerful and complete, are worth a look at in a second article. Aqua is more than more than pretty buttons and colors, it's a way of taking a powerful, yet obtuse OS, (Yes, Unix may be user friendly, but it's picky about who its friends are), and giving that power to everyone, not just the experts. There are things in Mac OS X that are no different than in X-Windows, Gnome, etc. The difference is, it's friendlier, and the OS tells you what is going on. The error messages are friendlier than OS 9's and far more friendlier than standard Unix. It does what it is supposed to do, and for the most part gets out of your, (well at least my), way while doing it. Yes it looks and acts different, but at heart, it's still the Mac OS, just in better shape, and with a new suit. Its kind of like Steve Austin after the bionics, faster, stronger, better.

Well, we covered a lot, and there's a lot to go. I didn't get to Classic, as that isn't something that is workable for me right now, and it's also been covered in depth. I only touched lightly on the Unix underpinnings, as I just haven't had a lot of time to really play with them. Hopefully, you have a better idea of how and maybe even why some of Mac OS X works the way it does. Remember, a lot of this can, and possibly will change, so if I seem to have avoided certain areas, that's why. Also, there is not a lot of administration-type documentation, which is a weakness with a lot of Apple products. OS X Server is a prime example. The Mac OS X Admin list from Omni Group is a better source of information than Apple for that product, and that's not right. Mac OS X is a product that is going to open up too many doors for Apple to be lackadaisical with Administration documentation. Whatever mistakes Microsoft and IBM make, they have always made sure the people running their products have really excellent support, as well as the people creating product for them. In Apple's defense, they do have the correct attitude, and with a little more documentation, they can make sure that the admins of the world are as well prepared for this OS as the developers.

In any event, Mac OS X is sure to be a wild ride between now and the full release, and this is only the first of many articles that you'll see in MacTech to help you make the ride as smooth as possible.

Bibliography and References

While I have no printed sources as such, I have to thank many sources, namely the folks on the OS X — admin list, who provided the NFS mounting information, Chuck Goolsbee's Mac — Manager list, MacFixIt, and all the moderators, and regular posters who make that such a useful sight, Dave Every of MacKiDo who has explained much about interfaces and other such technical things in a clear concise manner, Cal Simone, and Sal Soghoian, who helped me see that AppleScript was even more powerful than even I thought, especially in OS X, and many, many folks at other companies who will remain nameless, but have, without breaking any rules, managed to point me in the right direction time and time again.

MT

By Andrew C. Stone

Recursion: The Programmer's Friend

A programmer's bag of tricks is loaded with heuristics, algorithms and techniques, but of these, few are as powerful as recursion. The Oxford English Dictionary recursively defines recursion as "The application or use of a recursive procedure or definitiion"! Recursive has the definition "Involving or being a repeated procedure such that the required result at each step except the last is given in terms of the result(s) of the next step, until after a finite number of steps a terminus is reached with an outright evaluation of a result."

In software, recursion is when a function or method calls itself, over and over, with slightly differing arguments. Of course this sounds like the perfect recipe for an infinite loop, but we will design in an exit condition so you don't end up in Cupertino! Recursion allows the writing of elegant and terse code once you understand how it works.

Recursion abounds in nature, and can be visualized by thinking of the fractal Mandelbrot set — no matter how deep into the set you continue to go, the same forms appear over and over, in an increasingly minute, yet perfectly replicated form.

Programmers often use a recursive data structure called a **tree** to represent hierarchical data. A tree is a root (Now that's an oxyomoron!) with zero or more subtrees. Each subtree consists of a root node with zero or more subtrees. A subtree node with no branches or children is a leaf node. (Tree terminology uses both botanical (root/branch) and geneological (parent/child) terms.) A classic use of recursion is for tree traversal, where you want to perform some action on each node in the tree.

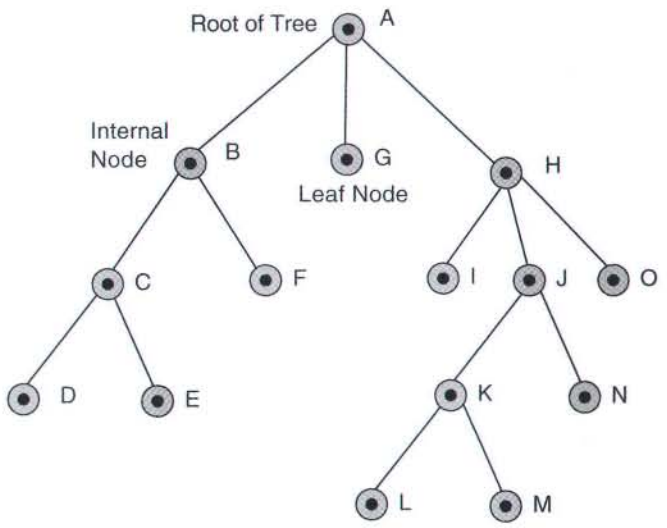


Figure 1. *A Tree with nodes labeled postorder.*

A tree can be implemented in various ways, depending on the structure and use of the tree. It's beyond the scope of this article to cover the implementation of "scales well to large N" trees such as the btree; however for a reasonably small number (e.g. under 1000) of nodes, arrays of arrays will work fine. Let's define a simplistic Node as:

```
@interface Node {
        id data; // an opaque pointer to some kind of data
        NSArray *_children;
    // if an internal node, this contains children nodes
    // if this is a leaf node, it contains 0 elements.
}
// query the Node:
- (NSArray *)children;
- (NSData *)data;
```

Andrew Stone <andrew@stone.com> is founder of Stone Design Corp <http://www.stone.com/> and has spent 12 years writing Cocoa software in its various incarnations.

```
// establish the Node:
- (void)setChildren:(NSArray *)newKids;
- (void)setData:(id)data;
@end

And our tree controller object would look like:

@interface TreeController {
    Node * _rootOfTreeNode;   // the root is all you need to get at any Node
}
- (void)visitNode:(Node *)node;
- (void)printData:(Node *)node;
@end
```

Walking a Tree: Kinds of Traversals

There are three types of tree traversals: **preorder**, **inorder**, and **postorder** - each defines the particulars of whether you work on a node before or after working on its children. In **preorder**, you work on the parent and then do an **inorder** traversal of each of the children . In **inorder**, you do a preorder traversal on the leftmost child, work on the parent, and then do an inorder traversal of each of the remaining children. In **postorder**, you do a postorder traversal of each of the children and then work on the parent.

Here's the code for a recursive preorder traversal:

```
- (void)visitNode:(Node *)node {
        NSArray *childrenToVisit = [node children];
        unsigned i, count = [childrenToVisit Count];

    // visit the current node:
        [self printData:[node data]].

    // if there are no children, then recursion ends:
        for (i = 0; i < count; i++) // make recursive call:
  [self visitNode:[childrenToVisit objectAtIndex:i]];
}

- (void)printData:(Node *)node {
    // this is just an example - in reality you might do something useful!
        NSLog([node description]);
}
```

To traverse an entire tree, you would simply start the recursive process at the top of the tree by calling our method with the root node as the argument:

```
[self visitNode:_rootOfTreeNode];
```

Note that to perform a postorder traversal, you just move the "printData" (our placeholder for the action we wish to perform on any node) to the bottom of the method:

```
- visitNodePostorder:(Node *)node {
        NSArray *childrenToVisit = [node children];
        unsigned i, count = [childrenToVisit Count];
        for (i = 0; i < count; i++)
  [self visitNodePostorder:[childrenToVisit objectAtIndex:i]];
    // now visit the current node:
        [self printData:[node data]].
}
```

Stacks and Stacks of Stacks

So how does this all work? Every programming language that supports recursion (including Objective-C) maintains a stack of information about parameters and local variables for each time a procedure, function, or method is called. When a procedure is called, the information is pushed onto the stack; when the procedure exits, the information is popped from the stack.

Given the tree pictured in Figure 1, let's mentally walk through what's happening. First we call visitNodePostorder: on the root node (node A), placing this method call on the stack. Node A has 3 children so we call visitNodePostorder on the first child (node B), pushing that call on the stack. Node B has 2 children, so we call visitNodePostorder on the first child (node C), placing that call on the stack. We then call the method on node C's first child, node D. Now the stack is 4 deep, but this final node is a leaf, so we call printData on node D. We're back in visitNodePostorder:C and the second child (node E) is called, increasing the stack to 4 again. Since it is also a leaf, we print it, pop the method call, and we're back down to 3. Now node C will print itself, and we're down to 2 deep in the stack. The second child of node B is a leaf, so we go 3 deep, and after printing, we're at 2 deep. Node B is then printed, and then we're back to the first frame in the stack (node A), on the second iteration of its for loop. Since the second child (node G) of the root node is a leaf, it gets printed, and we're on the third child. And so on for the rest of the tree.

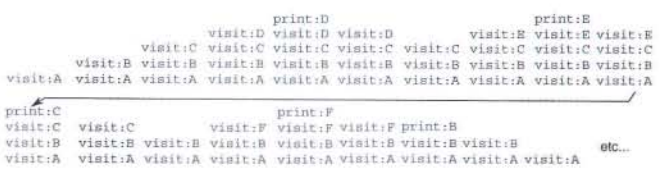


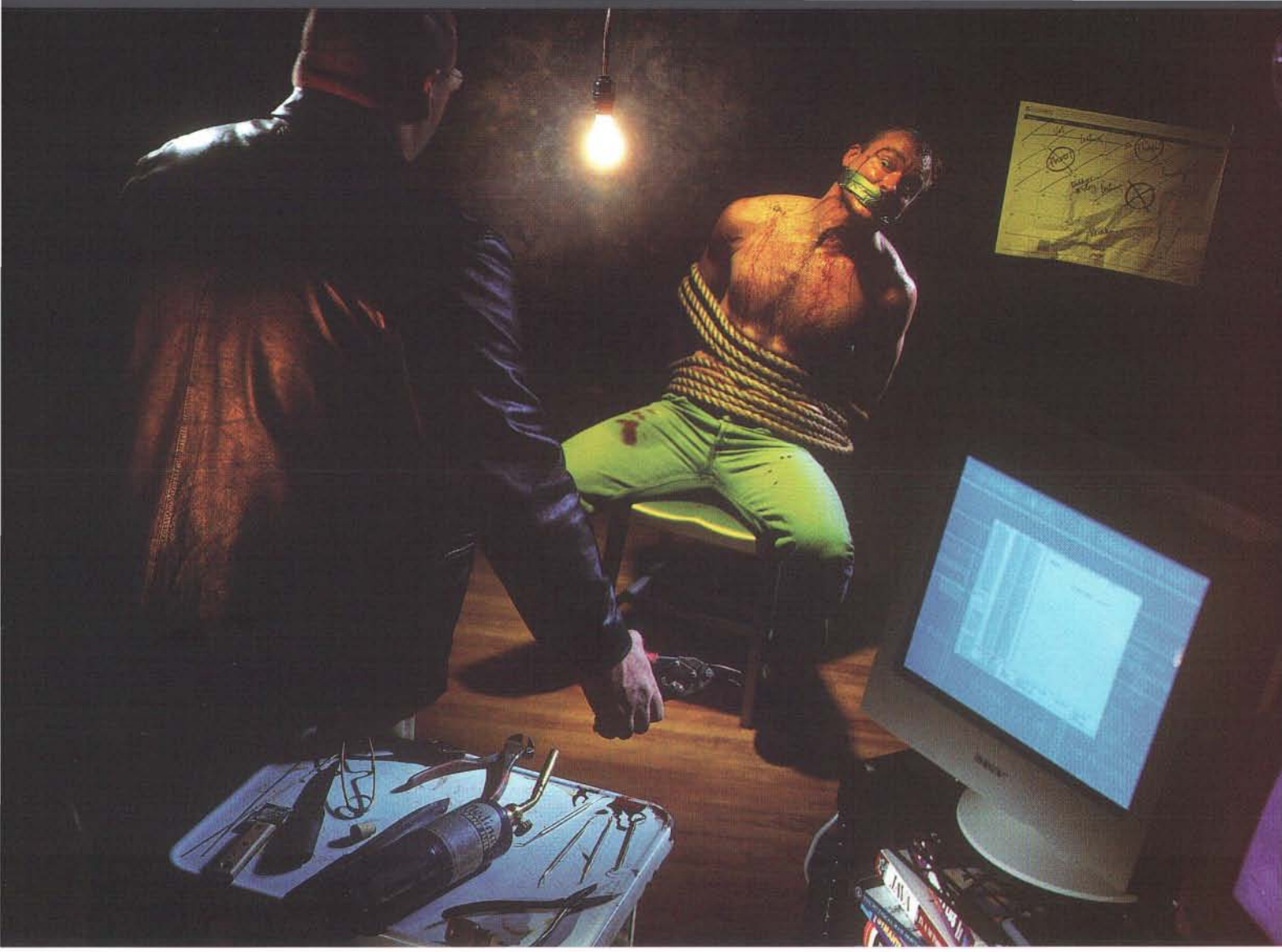
Figure 2. *Snapshots of the stack during the tree traversal.*

So, you can see that the stack will grow to the depth of the tree. If you are working with very deep trees and you're concerned with implementation efficiency and want to avoid the overhead inherent in making method calls, you can remove the recursion by managing your own stack of variables. However, this results in much more complex, less elegant, less maintainable code. I'd avoid it in all but the most extreme situations.

Conclusion

As programming challenges arise, remember this recursive jewel I learned many years ago in Computer Science school: if you cannot tackle a problem, divide it into smaller problems which you can tackle. Eventually you'll have wittled the problem down to something manageable, and applying this recursive problem solving tool will usually result in a readable and maintainable software solution.

MT



Programming Doesn't Have To Be This Difficult. It's REALbasic!

NEW Version! REALbasic 2.1* FREE Update!

REALbasic is the award-winning, visual, object-oriented BASIC development environment for the Macintosh.

Use REALbasic's visual interface builder and platform-independent language to build native, compiled—not interpreted—professional quality applications in a fraction of the time it would take in C/C++. Because our language is platform-independent you only need to write a single set of code to create applications for both Macintosh and Windows. Leverage your C/C++ experience to extend REALbasic's capabilities using shared libraries, Mac OS Toolbox and Win32 API calls or by writing cross-platform REALbasic plug-ins.

Create prototypes, Internet applications, database front-ends, even games with REALbasic! Its OOP language employs everything you'd expect from a modern development environment—methods, properties, classes, subclassing, inheritance, constructors and destructors, virtual methods, and more. The IDE supports multi-threading, extensibility, TCP/IP controls, plus multimedia and QuickTime tools, and support for standards such as SQL, ODBC, AppleScript, and XCMDs. You can even import Visual Basic forms and modules.

Go to www.realbasic.com NOW to download a FREE trial version or call 512.263.1233.

REALbasic®

MacPower 2000 Products of The Year

MacHome

Macworld

Apple Design Award

2000 Runner-Up - Best Macintosh User Experience

Mac Fan MVP '99 MOST VALUABLE PRODUCT

MacAddict RATED

*Free update for all owners of REALbasic 2.0 and above. REALbasic and the REALbasic logo are trademarks of REAL Software, Inc. Apple and the Apple logo are trademarks of Apple Computer, Inc., registered in the U.S., used with permission. All other trademarks are the property of their respective owners.

by Tim Monroe

Word Is Out

Using Text in QuickTime Movies

Introduction

When QuickTime was first introduced, it was able to handle two types of media data: video and sound. Curiously, the very next media type added to QuickTime (in version 1.5) was *text*, or the written word. Part of the motivation for adding text media was to provide the sort of "text below the picture" that you see in movie subtitles or television closed-captioning, as illustrated in **Figure 1**. Here, the text provides the words of a song, which can be useful to hearing-impaired or non-English speaking users. Similarly, the text might provide the dialogue of a play or a readable version of the narration. Of course, the text doesn't have to just mirror the voice part of an audio track; it can be any annotation that the movie creator deems useful for the viewer.

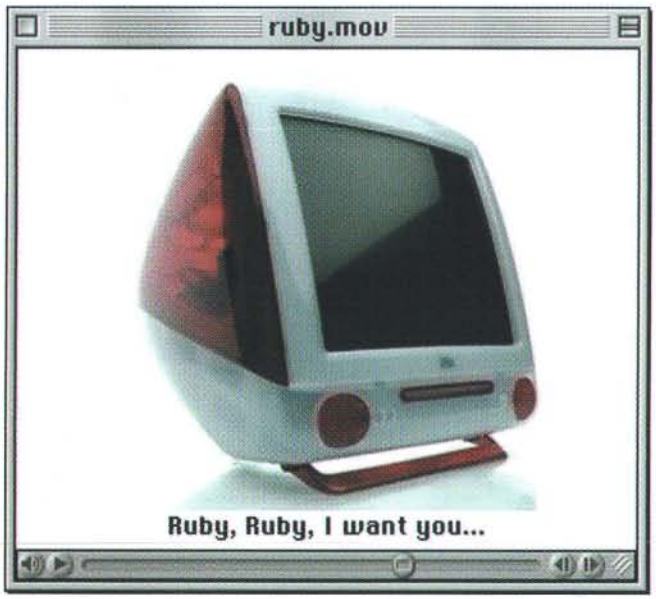


Figure 1: *A movie containing a text track.*

The text you see in **Figure 1** is not part of the video track; rather, it is stored in a *text track* (whose associated media is of type TextMediaType). Typically the text track is situated below the video track (as in **Figure 1**), but in fact it can overlay part or all of the video track. In order for both the text and the overlain video to be visible, the background of the text track should be transparent or "keyed out"; the text is then called *keyed text*. **Figure 2** shows some keyed text overlaying a video track. Keying can be computationally expensive, however, so keyed text is seen less often than below-the-video text.

Figure 2: *A movie containing a keyed text track.*

Tim Monroe works in the QuickTime Engineering group at Apple. You can contact him at monroe@apple.com.

QuickTime provides the capability to search for a specific string of characters in a text track and to move the current movie time forward (or backward) to the next (or previous) occurrence of that string. In addition, the standard movie controller provides support for a special kind of text track called a chapter track. A *chapter track* is a text track that has been associated with some other track (often a video or sound track); when a movie contains a chapter track, the movie controller will build, display, and handle a pop-up menu that contains the text in the various samples in that track. The pop-up menu appears (space permitting) in the controller bar. The various parts of the associated track are called the track's *chapters*. When the user selects an item in the pop-up menu, the movie controller jumps to the start time of the selected chapter. **Figure 3** shows our standard appearing-penguin movie with a chapter track that indicates the percentage of completion (both before and after the user clicks on the pop-up menu). Notice that we've had to hide the step buttons in the controller bar to make room for the chapter pop-up menu. Notice also that the text track itself is not visible.

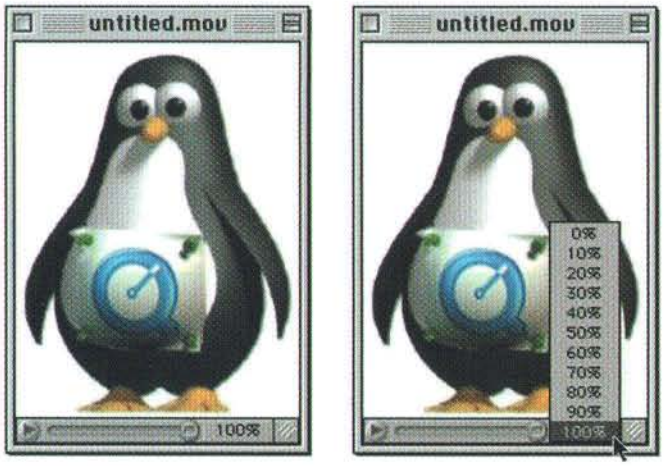


Figure 3: *A movie with a chapter track.*

The QuickTime Player application, introduced with QuickTime 3.0, employs a slightly different user interface for accessing a movie's chapters. As you can see in **Figure 4**, a QuickTime Player movie window replaces the pop-up menu with a set of up- and down-arrow controls, which select the previous and next chapter.

Get the power and experience you need from

PROSOFT
engineering, inc.

With over thirteen years in providing programming service to the Mac community. Prosoft is committed in delivering the ultimate quality software products and service.

Got OS X?

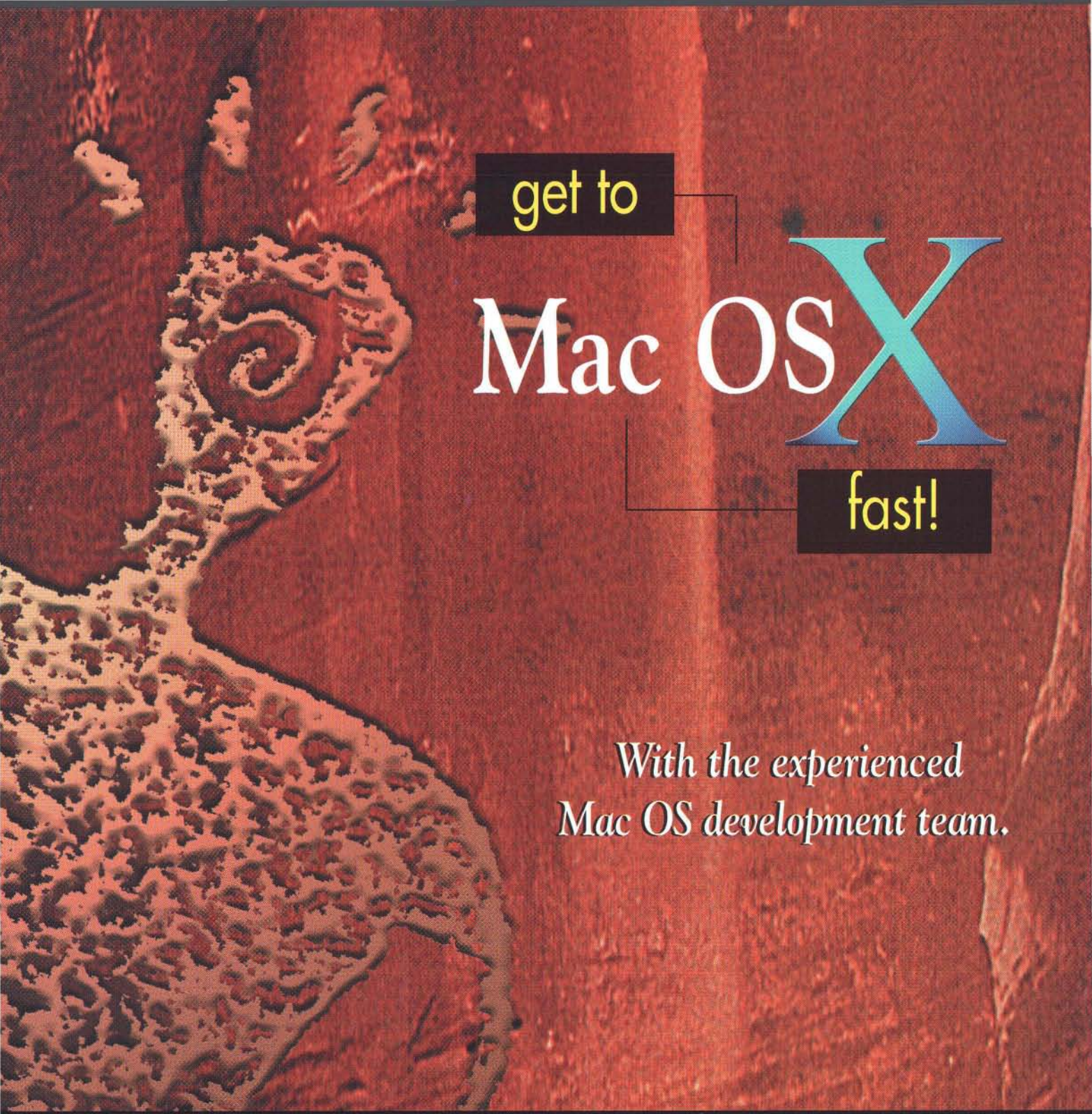
Our Engineers have extensive knowledge in:

- Macintosh PPC Drivers, Shims and Extensions
- Macintosh Power Plant Applications, Mac OS X applications and drivers
- Carbon application porting for Mac OS X
- Multimedia and QuickTime Applications
- Unix/Linux Console Applications, including Unix Solaris, xBSD Drivers, and Linux Drivers

Other areas of expertise

- Windows 95, 98
- Window CE
- Windows NT, 2000
- Unix / Linux
- Java
- Embedded OS
- RDBMS
- Network Consulting

www.prosofteng.com • tel: (925) 426-6100 • sales@prosofteng.com



ed Rock Software specializes in the Macintosh software devel-
pment. Red Rock Software's team of senior engineers aver-
ges over 10 years of development experience on the
lacintosh platform. Our expertise and experience has gained
s the trust of companies like *Apple Computer, Iomega*
orporation and Sorenson Media.

you are looking to get your product compatible with Mac
S X quickly and with extreme quality, let our experts help.

- Aqua Interface Implementation
- OS X Carbon Porting
- OS X Native Cocoa Application Development

RED ROCK
software

call Red Rock at 888.689.3038
or visit us online at www.redrocksw.com

- [ ] Eat your vegetables.
- [ ] Exercise every day.
- [x] Port to Mac OS X.
- [ ] Call your mom.

All of these are good for you. We can make one of them easy.

Since 1989, The Omni Group has worked with the technologies that have been refined into Mac OS X.

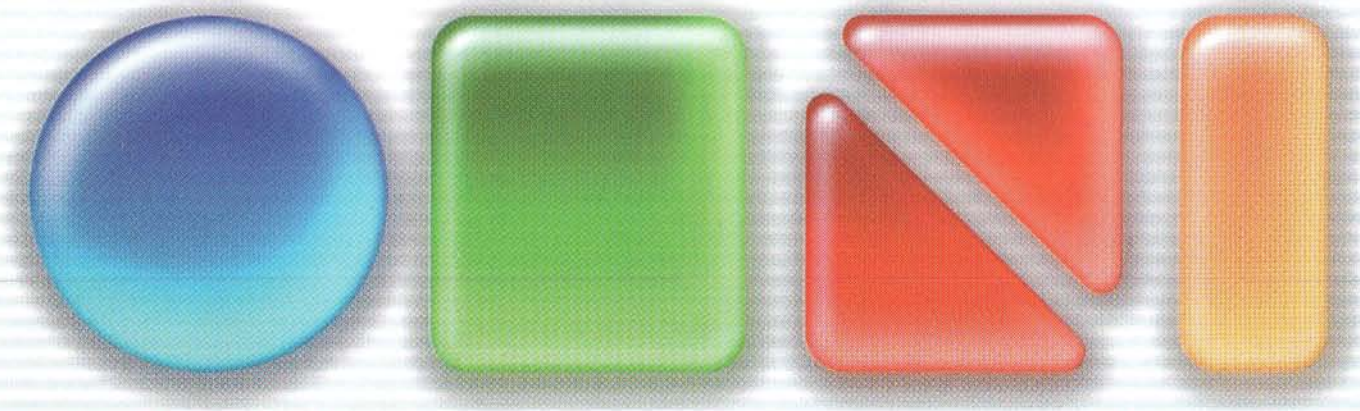
Moving to Mac OS X is a big step for your company, and you need consultants who can help you both plan how best to make the transition and follow whatever path you choose.

That's been our business for years. No matter what kind of product you have, we can get it up under OS X, fast:

- Real games: We ported id's Doom and Quake games to NEXTSTEP and OS X. Quake 2 took us a week.
- Big apps: We ported Adobe's FrameMaker to NEXTSTEP and Sun's Concurrence to OpenStep/Solaris.
- Big libraries: We ported the Oracle 8 client libraries which Apple ships today in OS X Server.
- Serious drivers: We ported 3dfx's Glide and wrote Voodoo2 and Rendition drivers for OS X Server. We've written new mouse drivers for OS X Server and joystick drivers for OpenStep.
- New apps: We wrote OmniWeb, the only native OS X web browser, and OmniPDF, the native Acrobat viewer for OS X.

Mac OS X is what we do. Let us help you do it, too.

The Omni Group



2707 Northeast Blakeley Street
Seattle, Washington 98105-3118
www.omnigroup.com/consulting

sales@omnigroup.com
800.315.OMNI x201
206.523.4152 x201

makers of

PERIPHERALS, EXPANSION CARDS & SMART DEVICES

get OS X compatible

Since 1991, high tech companies like Apple Computer, Hewlett Packard, and Leica Microscopy have turned to Art & Logic for assistance in implementing cutting edge technologies.

Now, with the advent of OS X, Art & Logic is helping companies make their products compatible with Apple's new operating system. Art & Logic engineers are industry leaders in Carbon & Cocoa porting, Aqua implementation, and driver development.

In the new economy, where time to market is everything, you need results. Art & Logic delivers.

"We have found the engineers at Art & Logic to be outstanding—better than excellent. When we use their software development services, we get results."

Ted Dykes, CEO of LRO, Inc.

Art & Logic, Inc.
www.artlogic.com
877-278-5644 (toll free)
info@artlogic.com

The software engineering company that helps bring your hardware product to market—guaranteed.

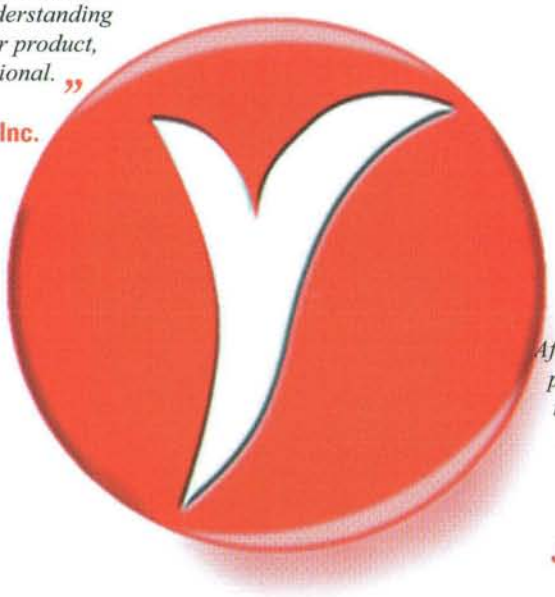
Robosoft has grown to be one of the most experienced Mac product development outsourcing companies. Here are a few reasons why ...

"*Robosoft's best of class development practices help them deliver the impossible -- **high quality solutions, on budget, on time.** They are one of the finest Mac developers we have in Asia and Apple is proud to be their partners in progress.*"

Eshwar Vangala, Apple Computers.

"*Robosoft completely rewrote my software application exceeding my every expectation. In terms of **attention to technical detail** and understanding the business issues of the NetJumper product, the Robosoft team were exceptional.*"

Gilbert Borman, NetJumper Inc.



"*After working with Robosoft on Mac VersaBook project, we have no doubts that we will want to continue working with them in the future. Their **attitude was always positive** and did anything that we needed to help us get the product released.*"

Jake Benjamin, Versaware Technologies.

robosoftin.com

"*Robosoft Technologies receives my very highest recommendation. Prior to awarding Robosoft my current ASP project, I had never met the staff and had great concerns about distance and communications. My worries turned out to be unfounded. Robosoft could not have been **more responsive** if they were in the next office.Although I have yet to meet them personally, I think of Robosoft, not as a contractor, but as my IT partner.*"

Barry Shapiro, The Anderson Group.

Partners in Product Development

info@robosoftin.com
+91>8252>35458

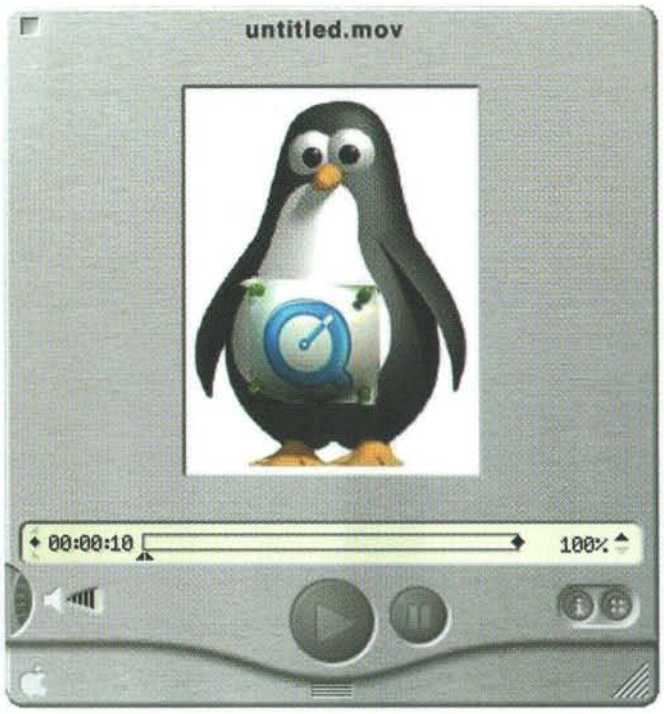


Figure 4: *The chapter controls in a QuickTime Player movie window.*

QuickTime 3.0 also included a web browser plug-in that supports *linked text*. Linked text is contained in a *hypertext reference track* (usually shortened to *HREF track*), which is simply a text track that has a special name (to wit, "HREFTrack") and contains some media samples that pick out URL links. If a text sample contains text of the form *<URL>*, the QuickTime Plug-In will load the specified URL in the frame containing the movie when the user clicks in the movie box while that text sample is active. (Let's call this a *clickable link.*) If the text is if the form A*<URL>*, then the plug-in will load the specified URL automatically when that text sample becomes active. (Let's call this an *automatic link.*)

QuickTime 4 added one more text-handling tool, the ability to attach wired actions to data in a text track. A *wired action* is some action (such as setting a movie's volume or its current time) that is initiated by some particular event. The events that can trigger wired actions include both user events like moving or clicking the mouse and movie controller events like loading movies or processing idle events from the operating system. We'll investigate wired actions at length in a future article; for the moment, consider the movie shown in **Figure 5**. This movie contains only one track, a text track. The text track is configured so that clicking on the word "Apple" launches the user's default web browser and loads the URL http://www.apple.com; in addition, rolling the cursor over the word "CNN" loads the URL http://www.cnn.com/. (Let's call this *wired text.*)

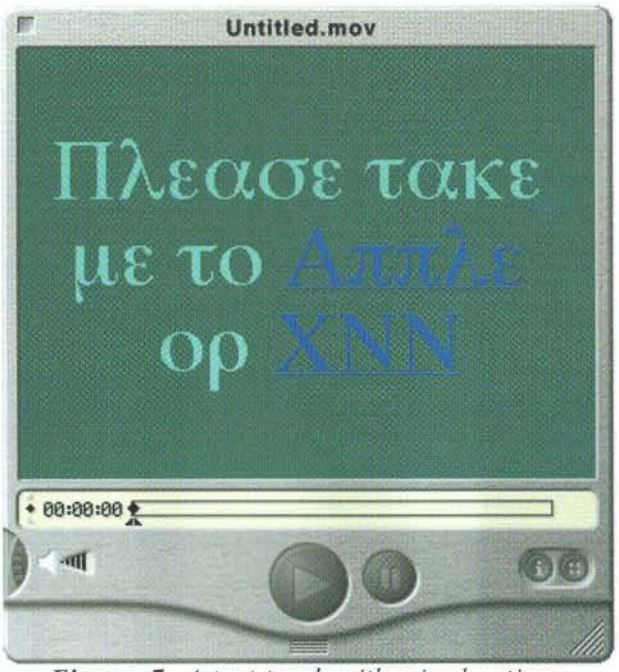


Figure 5*: A text track with wired actions.*

In this article, we're going to take a look at the most basic ways of handling text in QuickTime movies. After we take a brief detour to upgrade the code that adjusts our Edit menu, we'll uncover some ways in which our existing sample applications can already interact with text. It turns out that these applications can do a surprising amount of work with text tracks; indeed, they can even create text tracks, in spite of the fact that they contain no text-specific code. So we'll spend a little bit of time to see how that's possible. Then we'll see how to create text tracks using the standard Movie Toolbox functions. We'll also learn how to search and edit text tracks. Toward the end of this article, we'll see how to create chapter tracks and HREF tracks. When all is said and done, we'll have at hand the essential tools that we need to create text tracks, keyed text, chapter tracks, and linked text. **Figure 6** shows the Test menu of this month's sample application, named QTText.

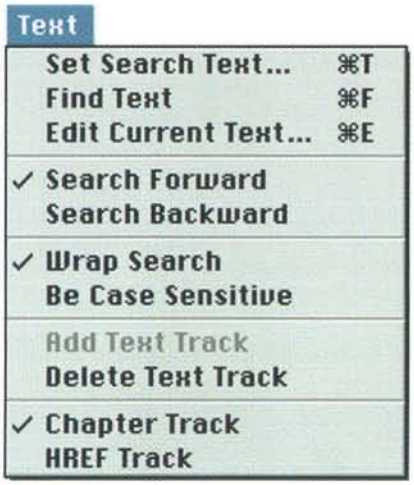


***Figure 6**: The Test menu of QTText.*

The Edit Menu Revisited

Let's begin by considering our code for enabling and disabling items in the Edit menu. (This might appear to have nothing at all to do with text handling, but it is actually fairly germane to this topic. Trust me.) Currently, when the user clicks in the menu bar to select one of our application's menus, our application framework calls the QTFrame_AdjustMenus function. (In our Macintosh framework, this happens in response to a mouseDown event in the inMenuBar window part; in our Windows framework, this happens when the MDI frame window receives the WM_INITMENU command.) Listing 1 shows the code in QTFrame_AdjustMenus that adjusts the Edit menu.

Listing 1: Adjusting the Edit menu (original version)

QTFrame_AdjustMenus

```
#if TARGET_OS_MAC
myMenu = GetMenuHandle(kEditMenuResID);
#endif
if (myMC != NULL) {
  long      myFlags;

  MCGetControllerInfo(myMC, &myFlags);

  QTFrame_SetMenuItemState(myMenu, IDM_EDITUNDO,
                    myFlags & mcInfoUndoAvailable ?
                kEnableMenuItem : kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITCUT,
                    myFlags & mcInfoCutAvailable ?
                kEnableMenuItem : kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITCOPY,
                    myFlags & mcInfoCopyAvailable ?
                kEnableMenuItem : kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITPASTE,
                    myFlags & mcInfoPasteAvailable ?
                kEnableMenuItem : kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITCLEAR,
                    myFlags & mcInfoClearAvailable ?
                kEnableMenuItem : kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITSELECTALL,
                    myFlags & mcInfoEditingEnabled ?
                  kEnableMenuItem : kDisableMenuItem);
QTFrame_SetMenuItemState(myMenu, IDM_EDITSELECTNONE,
                    myFlags & mcInfoEditingEnabled ?
                kEnableMenuItem : kDisableMenuItem);
} else {
  QTFrame_SetMenuItemState(myMenu, IDM_EDITUNDO,
```

```
    QTFrame_SetMenuItemState(myMenu, IDM_EDITUNDO,
                             kDisableMenuItem);
    QTFrame_SetMenuItemState(myMenu, IDM_EDITCUT,
                             kDisableMenuItem);
    QTFrame_SetMenuItemState(myMenu, IDM_EDITCOPY,
                             kDisableMenuItem);
    QTFrame_SetMenuItemState(myMenu, IDM_EDITPASTE,
                             kDisableMenuItem);
    QTFrame_SetMenuItemState(myMenu, IDM_EDITCLEAR,
                             kDisableMenuItem);
    QTFrame_SetMenuItemState(myMenu, IDM_EDITSELECTALL,
                             kDisableMenuItem);
    QTFrame_SetMenuItemState(myMenu, IDM_EDITSELECTNONE,
                             kDisableMenuItem);
}
```

There's nothing particularly complicated here: if there is no movie controller associated with the frontmost window or there is no frontmost window, then we disable *all* the items in the Edit menu (that's the "else" portion). Otherwise, we call the MCGetControllerInfo function to determine the current status of the movie controller and its associated movie. MCGetControllerInfo returns a set of flags that indicate which editing operations currently make sense for the specified movie controller and its movie. For instance, if there is some data available for pasting and editing is enabled, then the mcInfoPasteAvailable flag is set in the 32-bit long integer returned by MCGetControllerInfo. In this case, our application should enable the Paste menu item. Conversely, if either editing is disabled for the specified movie or there is nothing to paste, then that flag is clear. In that case, the Paste menu item should be disabled. We call the function QTFrame_SetMenuItemState to enable or disable the Paste menu item, like this:

```
QTFrame_SetMenuItemState(myMenu, IDM_EDITPASTE,
                myFlags & mcInfoPasteAvailable ?
            kEnableMenuItem : kDisableMenuItem);
```

We've already considered QTFrame_SetMenuItemState in an earlier article (see "QuickTime 101" in *MacTech*, January 2000); it just calls the appropriate platform-specific function for enabling or disabling a menu item.

Emulating QuickTime Player

So far, so good. But there is a very important capability that we still need to add to our sample applications. If we launch the QuickTime Player application, open a movie, make a selection, and then hold down the Option key (or, on Windows, both the Ctrl and Alt keys) while clicking on the Edit menu, we'll see something like the menu shown in **Figure 7**:

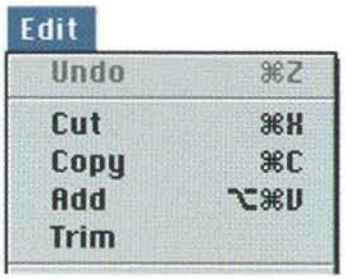


***Figure 7**: The Edit menu of QuickTime Player (Option key down).*

Notice that the Paste menu item is now labeled "Add" and the Clear menu item is now labeled "Trim". Similarly, if we hold down just the Shift key while clicking on the Edit menu, we'll see the menu shown in **Figure 8**:

***Figure 8**: The Edit menu of QuickTime Player (Shift key down).*

Now the Paste menu item is labeled "Replace". Finally, if we hold down the Option and the Shift keys (or, on Windows, the Ctrl and Alt and Shift keys) while clicking on the Edit menu, the Paste menu item will be labeled "Add Scaled", as shown in **Figure 9**. (For the moment, don't worry about what these renamed menu items actually *do*; we'll get to that in the next section.)

***Figure 9**: The Edit menu of QuickTime Player (Shift and Option keys down).*

What's happening here is that QuickTime Player is not using MCGetControllerInfo to do its Edit menu

adjusting, at least for the first five menu commands. Instead, it's using the MCSetUpEditMenu function, which is specially designed to change the Edit menu item labels in the ways just described, depending on which keyboard modifier keys the user is holding down. MCSetUpEditMenu is declared essentially like this:

```
ComponentResult MCSetUpEditMenu (MovieController mc,
                    long modifiers, MenuHandle mh);
```

MCSetUpEditMenu correctly enables or disables and names the first five commands in the Edit menu specified by the menu handle mh, as long as those items have the standard arrangement (Undo, a separator line, Cut, Copy, Paste, and Clear).

It appears, then, that we can simplify our menu-adjusting code and gain the additional behaviors described above by using MCSetUpEditMenu ourselves. There are just a couple of changes we need to make to support MCSetUpEditMenu. Primarily, we need to add a parameter to our QTFrame_AdjustMenus function, so that we can pass it the current keyboard modifiers. Henceforth, QTFrame_AdjustMenus will be declared like this:

```
int QTFrame_AdjustMenus (WindowReference theWindow,
            MenuReference theMenu, long theModifiers);
```

Getting the appropriate keyboard modifiers in our Macintosh code is easy. Whenever we call QTFrame_AdjustMenus, either we don't care about the modifiers (so we can pass 0L) or we have an event record available (so we can pass (long)theEvent->modifiers).

Getting the Modifier Keys on Windows

When we call QTFrame_AdjustMenus on Windows, however, we need to do some additional work to determine which (if any) modifier keys the user is holding down when clicking on the Edit menu. Remember that we want to pass MCSetUpEditMenu a long integer whose bits indicate which modifier keys are active. The "gotcha" here is that these are supposed to be the modifier keys on a *Macintosh* keyboard. MCSetUpEditMenu knows nothing about the Alt or Ctrl keys found on Windows keyboards. Rather, it's expecting a 32-bit value in which the up or down state of the relevant modifier keys is encoded using these bits (defined in Events.h):

```
enum {
  cmdKey          = 1 << cmdKeyBit,    // 0x0100
  shiftKey        = 1 << shiftKeyBit,  // 0x0200
  alphaLock       = 1 << alphaLockBit, // 0x0400
  optionKey       = 1 << optionKeyBit,          // 0x0800
  controlKey      = 1 << controlKeyBit// 0x1000
};
```

For example, if only the Option key is down, the modifiers value should be 0x00000800. Similarly, if both the Shift and Control keys are down, the modifiers value should be 0x00001200.

QuickTime maps the Windows modifier keys to the Macintosh modifier keys in this manner:

- The Windows Alt key is mapped to the Macintosh Control key.
- The Windows Ctrl key is mapped to the Macintosh Command key.
- The Windows Shift key is mapped to the Macintosh Shift key.
- The Windows Caps Lock key is mapped to the Macintosh Caps Lock key.
- The combination of the Windows Alt and Ctrl keys is mapped to the Macintosh Option key.

To help us construct a Mac-style modifiers long word, we'll add these constants and compiler macros to the file WinFramework.h:

```
#define VK_MAC_CONTROLKEY      VK_MENU
#define VK_MAC_COMMANDKEY      VK_CONTROL
#define VK_MAC_SHIFTKEY        VK_SHIFT
#define VK_MAC_CAPSKEY       VK_CAPITAL

#define QTFrame_IsControlKeyDown(theKeyState)
      (theKeyState[VK_MAC_CONTROLKEY] & 0x80 ? 1 : 0)
#define QTFrame_IsCommandKeyDown(theKeyState)
      (theKeyState[VK_MAC_COMMANDKEY] & 0x80 ? 1 : 0)
#define QTFrame_IsShiftKeyDown(theKeyState)

        (theKeyState[VK_MAC_SHIFTKEY] & 0x80 ? 1 : 0)
#define QTFrame_IsAlphaLockKeyDown(theKeyState)
        (theKeyState[VK_MAC_CAPSKEY] & 0x80 ? 1 : 0)
#define QTFrame_IsOptionKeyDown(theKeyState)
            (QTFrame_IsControlKeyDown(theKeyState)) &&
            (QTFrame_IsCommandKeyDown(theKeyState))
```

On Windows, a *key state array* (represented by the argument theKeyState in these macros) is a 256-byte array that contains information about each of the 256 virtual-key codes. If a key is down, then the high-order bit (0x80) of the corresponding element of this array will be set. For instance, if the Alt key is down, then the high-order bit of the array element whose index is 0x12 will be set. (The virtual-key code for the Alt key is VK_MENU, which is defined as 0x12 in the file Winuser.h.)

We can fill a key state array with the current values by calling the GetKeyboardState function. Then all we need to do is inspect the Windows modifier keys that are of interest to us and construct a Mac-style modifiers value that encodes that information. When we need to call QTFrame_AdjustMenus, we can get the current set of modifier keys by calling QTFrame_GetKeyboardModifiers, defined in Listing 2.

Listing 2: Getting the Windows keyboard modifier keys

QTFrame_GetKeyboardModifiers

```
static long QTFrame_GetKeyboardModifiers (void)
{
  long  myModifiers = 0L;
  BYTE  myKeyState[256];

  if (GetKeyboardState(&myKeyState[0])) {
    if (QTFrame_IsOptionKeyDown(myKeyState))
      myModifiers |= optionKey;
    else if (QTFrame_IsCommandKeyDown(myKeyState))
      myModifiers |= cmdKey;
    else if (QTFrame_IsControlKeyDown(myKeyState))
      myModifiers |= controlKey;

    if (QTFrame_IsShiftKeyDown(myKeyState))
      myModifiers |= shiftKey;
    if (QTFrame_IsAlphaLockKeyDown(myKeyState))
      myModifiers |= alphaLock;
  }

  return(myModifiers);
}
```

So, on Windows, we are now able to pass the correct set of modifier flags to MCSetUpEditMenu. But what do we pass for the third parameter, which on MacOS is a menu handle for the Edit menu? The answer, it turns out, is that we'll pass the value NULL. The reason for this is that on Windows we access our menus using a value of type HMENU, not MenuHandle. This means, however, that on Windows we cannot depend on MCSetUpEditMenu to either highlight or rename the items in the Edit menu. For that, we'll have to write our own code.

Renaming the Edit Menu Items on Windows

At this point, you might be wondering why we're bothering to call MCSetUpEditMenu on Windows, if it isn't going to help us with highlighting or renaming the items in the Edit menu. The answer is that MCSetUpEditMenu does more than simply enable or disable menu items and rename them to match the state of the active modifier keys. MCSetUpEditMenu also sets some flags maintained internally by the movie controller that affect the operation of subsequent editing commands. For instance, when we call MCPaste, it looks at those flags to determine whether it should paste, or replace, or add, or add scaled. In other words, if we don't call MCSetUpEditMenu, all our editing operations will just be the default undo, cut, copy, paste, and clear operations.

On Windows, we still have two tasks left to handle. First, we need to perform our own Edit menu item enabling and disabling. We already have code for this (see Listing 1 again), so we'll just conditionalize that code to be executed under Windows but not under MacOS. Second, we need to find a way to rename the Edit menu items according to the current state of the modifier keys. This task is actually relatively easy, since QuickTime provides the MCGetMenuString function, which we can use to retrieve the label for a particular menu item, given a set of modifier keys. Suppose, for instance, that we execute this line of code (here, myString is of type Str255):

```
MCGetMenuString(myMC, optionKey, mcMenuPaste, myString);
```

If MCGetMenuString completes successfully, then myString will hold the string "Add". All we need to do then is insert that string into our Windows Edit menu. The function QTFrame_ConvertMacToWinMenuItemLabel, defined in

Listing 3, handles all of this for us.

Listing 3: Renaming a Windows Edit menu item

QTFrame_ConvertMacToWinMenuItemLabel

```
void QTFrame_ConvertMacToWinMenuItemLabel (
      MovieController theMC, MenuReference theWinMenu,
      long theModifiers, UInt16 theMenuItem)
{
  Str255   myString;
  char     *myLabelText = NULL;
  char     *myBeginText = NULL;
  char     *myFinalText = NULL;
  short    myLabelSize = 0;

  // get the appropriate label for the specified item and keyboard modifiers
  MCGetMenuString(theMC, theModifiers,
                  MENU_ITEM(theMenuItem), myString);

  switch (theMenuItem) {
    case IDM_EDITUNDO:
      myBeginText = kAmpersandText;
      myFinalText = kWinUndoAccelerator;
      break;
    case IDM_EDITPASTE:
      myBeginText = "";
      myFinalText = kWinPasteAccelerator;
      break;
    case IDM_EDITCLEAR:
      myBeginText = kAmpersandText;
      myFinalText = kWinClearAccelerator;
      break;
    default:
      // currently, only the Undo, Paste, and Clear items are modified by
      // MCSetUpEditMenu, so that's all we'll handle here
      return;
  }

  myLabelSize = strlen(myBeginText) + myString[0] +
                                      strlen(myFinalText) + 1;
  myLabelText = malloc(myLabelSize);
  if (myLabelText == NULL)
    return;

  BlockMove(myBeginText, myLabelText, strlen(myBeginText));
  BlockMove(&myString[1], myLabelText + strlen(myBeginText),
                  myString[0]);
  BlockMove(myFinalText, myLabelText + strlen(myBeginText) +
                  myString[0], strlen(myFinalText));
  myLabelText[myLabelSize - 1] = '\0';

  QTFrame_SetMenuItemLabel(theWinMenu, theMenuItem,
                                      myLabelText);

  free(myLabelText);
}
```

QTFrame_ConvertMacToWinMenuItemLabel also adds the ampersand (&) to the beginning of several of the Edit menu items (so that the first letter is underlined) and the appropriate keyboard accelerator label to the end of all of them.

Putting it All Together

We're finally ready to put all these pieces together. Listing 4 shows our revised version of Listing 1. When no movie controller is available, we disable all the Edit menu items in exactly the same manner we did in our earlier version. And for the "Select All" and "Select None" items, we call MCGetControllerInfo and QTFrame_SetMenuItemState, just like before. But for the five standard Edit menu commands, we now call MCSetUpEditMenu on both Mac and Windows. In addition, on Windows we need

to do all menu item enabling and disabling ourselves, and we need to update the menu item labels, using our function QTFrame_ConvertMacToWinMenuItemLabel.

Listing 4: Adjusting the Edit menu (revised version)

QTFrame_AdjustMenus

```
#if TARGET_OS_MAC
myMenu = GetMenuHandle(kEditMenuResID);
#endif

if (myMC == NULL) {
  // if there is no movie controller, disable all the Edit menu items
  QTFrame_SetMenuItemState(myMenu, IDM_EDITUNDO,
                                        kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITCUT,
                                        kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITCOPY,
                                        kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITPASTE,
                                        kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITCLEAR,
                                        kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITSELECTALL,
                                        kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITSELECTNONE,
                                        kDisableMenuItem);
} else {
  MCGetControllerInfo(myMC, &myFlags);

  QTFrame_SetMenuItemState(myMenu, IDM_EDITSELECTALL,
                      myFlags & mcInfoEditingEnabled ?
                      kEnableMenuItem : kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITSELECTNONE,
                      myFlags & mcInfoEditingEnabled ?
                      kEnableMenuItem : kDisableMenuItem);

#if TARGET_OS_MAC
  MCSetUpEditMenu(myMC, theModifiers, myMenu);
#endif
#if TARGET_OS_WIN32
  MCSetUpEditMenu(myMC, theModifiers, NULL);

  QTFrame_SetMenuItemState(myMenu, IDM_EDITUNDO,
                      myFlags & mcInfoUndoAvailable ?
                      kEnableMenuItem : kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITCUT,
                      myFlags & mcInfoCutAvailable ?
                      kEnableMenuItem : kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITCOPY,
                      myFlags & mcInfoCopyAvailable ?
                      kEnableMenuItem : kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITPASTE,
                      myFlags & mcInfoPasteAvailable ?
                      kEnableMenuItem : kDisableMenuItem);
  QTFrame_SetMenuItemState(myMenu, IDM_EDITCLEAR,
                      myFlags & mcInfoClearAvailable ?
                      kEnableMenuItem : kDisableMenuItem);

  QTFrame_ConvertMacToWinMenuItemLabel
              (myMC, myMenu, theModifiers, IDM_EDITUNDO);
  QTFrame_ConvertMacToWinMenuItemLabel
              (myMC, myMenu, theModifiers, IDM_EDITCUT);
  QTFrame_ConvertMacToWinMenuItemLabel
              (myMC, myMenu, theModifiers, IDM_EDITCOPY);
  QTFrame_ConvertMacToWinMenuItemLabel
              (myMC, myMenu, theModifiers, IDM_EDITPASTE);
  QTFrame_ConvertMacToWinMenuItemLabel
              (myMC, myMenu, theModifiers, IDM_EDITCLEAR);
#endif
}
```

There is one final modification that we need to make to our Windows source code. Apparently, on Windows, calling the MCIsPlayerEvent function has the nasty side-effect of clearing the movie controller flags that store the current modifier key settings. So we need to make sure that we do not call MCIsPlayerEvent if we are about to execute an editing command. We can do this by adding the condition (theMessage != WM_COMMAND) in the movie window procedure QTFrame_MovieWndProc. See the version of WinFramework.c included in this month's code for the exact placement of this fix.

Text Importing

Suppose now that we've implemented all the changes described in the previous section. Let's see what all this work has bought us.

Importing Text from the Clipboard

Open a movie with a video track, perhaps even the penguin movie we created in an earlier article. Select part or all of the movie. Then switch to some application that can handle text; in that application, select some text and copy it. Then return to our upgraded application and execute the "Add Scaled" command in the Edit menu (that is, choose Paste while holding down the Shift and Option keys on the Mac, or the Shift and Ctrl and Alt keys on Windows). Voilà — we've just added a text track to our movie, positioned below the video track.

Keep in mind that our upgraded sample applications contain absolutely no special code for handling text media. So how did we manage to create a text track so effortlessly? The answer is that MCPaste looks to see what kind of data it's being asked to insert into the open movie. If it's a segment of a movie, then MCPaste just inserts the data as we'd expect. But if the data isn't movie data, MCPaste looks around for a QuickTime component that can import that kind of data as a movie. In other words, MCPaste goes looking for a suitable movie import component. In this case, it finds the *text movie import component* (component type MovieImportType and subtype TextMediaType), which inspects the current modifier flags cached by the movie controller and performs the operation corresponding to those flags.

- If none of the relevant modifier flags is set, the text movie importer pastes the text data at the current position in the movie. If a text track already exists in the movie, the pasted text is inserted into that track and inherits all the spatial and visual characteristics of that track. But if no text track exists in the movie, the text movie importer creates a new track that has the same size and position as the current movie box. The pasted text is given a default duration of two seconds.

- If only the Shift modifier flag is set, then MCPaste performs a Replace operation. If the movie has a non-empty selection, the pasted text replaces the current selection; otherwise, if there is no selection, the pasted

text replaces the *entire* movie. In both cases, the duration of the pasted text sample is the default two seconds.

- If only the Option modifier flag is set, then MCPaste performs an Add operation: the text track is positioned below the existing video track, with a height that accommodates the pasted text (this is called *adding in parallel*). The duration of the pasted text sample is the default two seconds.

- If both the Shift modifier flag and the Option modifier flag are set, then MCPaste performs an Add Scaled operation: a text track is added in parallel for the duration of the current selection. If there is no selection, then the text track is added in parallel for the duration of the entire movie.

On Macintosh operating systems only, holding down the Control key and any other combination of modifier keys while selecting Paste in the Edit menu causes the text movie importer to display the *text import settings dialog box*, shown in **Figure 10**. This dialog box allows the user to configure some settings of the pasted text.

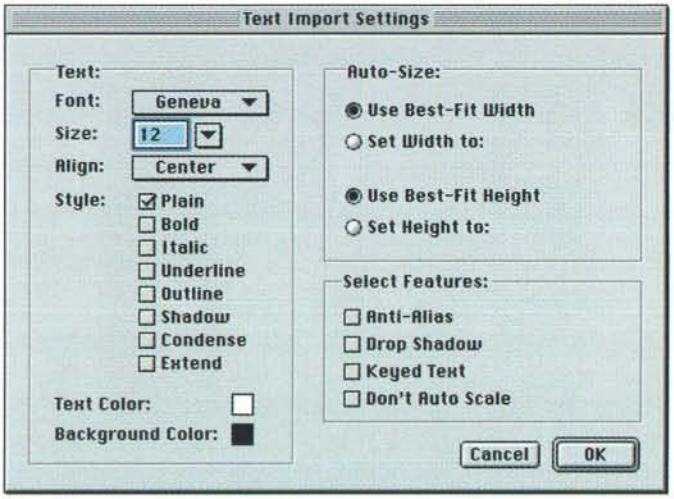


Figure 10: *The text import settings dialog box*

Importing Text from a File

The text movie importer can also import text stored in a file, and indeed provides some additional capabilities that are not available when pasting text from the scrap. If we open a text file using any of our sample applications, the text importer creates a movie that has a text sample for every paragraph of text in the file. Each text sample will have the standard default duration of two seconds

and will be drawn in the default text font, which is dependent upon the operating system.

Note that, since we're importing a file and not pasting data from the system scrap, our existing sample applications will exhibit this behavior, whether or not we've applied the changes described in the previous section. This is just another case of NewMovieFromFile detecting that the file we've asked it to process is not a QuickTime movie file and then looking around for a suitable movie importer to handle that data. (See "Quick on the Draw" in *MacTech*, April 2000 for more details on this.)

The text importer recognizes a large number of *text descriptors* that modify the default characteristics of the imported text. Suppose that we open a text file that contains these lines of text:

```
{QTtext}
{font:Tekton}{plain}{size:18}
{textColor: 0, 0, 0}{backColor: 65535, 65535, 0}
{justify:center}{timeScale:600}{width:240}{height:40}
{timeStamps:absolute}{language:0}{textEncoding:0}
{shrinkTextBox: on}
[00:00:00.000]
{textBox: 10, 0, 30, 240}We forgot to seed!
[00:00:01.000]
{textBox: 10, 0, 30, 240}D'Oh!
[00:00:01.100]
{textBox: 10, 20, 30, 240}D'Oh!
[00:00:01.200]
{textBox: 10, 40, 30, 240}D'Oh!
[00:00:01.300]
{textBox: 10, 60, 30, 240}D'Oh!
[00:00:01.400]
{textBox: 10, 80, 30, 240}D'Oh!
[00:00:01.500]
```

The text importer inspects the text descriptors found within the braces and creates the movie whose first frame is shown in **Figure 11**.

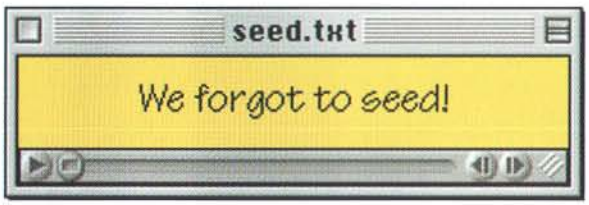


Figure 11: *The imported movie.*

Unfortunately, we don't have space to investigate text descriptors in more detail here. For complete documentation on using the available text descriptors, see the sources mentioned at the end of this article.

Text Tracks

As we've seen, the text movie importer provides our applications with a good deal of text-handling power at a very small cost. In fact, we didn't have to do anything at all to allow our applications to import text files, and we simply had to upgrade our Edit menu adjusting code to allow them to handle pasted text. But we still need to see how to create text tracks directly, without relying on the text movie importer. After all, we want to be able to work with text data that's not read from a file or from the system scrap.

Adding Text Media Samples

By this point in this series of articles, programmatically adding a track to a movie should be old-hat (since we've done this two or three times so far). We just need to call NewMovieTrack and NewTrackMedia to create a new track and media, call BeginMediaEdits to begin a media-editing session, call AddMediaSample to add samples to the media, call EndMediaEdits to end the media-editing session, and then call InsertMediaIntoTrack to insert the newly-edited media into the track. For any new kind of media that we encounter, we really need to ask only two questions: (1) what is the format of the data in the media samples? And, (2) what is the structure of the sample description that we need to pass to AddMediaSample?

For a text track, the media sample data is just the string of characters in the text itself, preceded by a 16-bit length field that specifies the number of characters in that string. And the appropriate sample description is a *text description structure*, defined by the TextDescription data type:

```
struct TextDescription {
  long                          descSize;
  long                          dataFormat;
  long                          resvd1;
  short                         resvd2;
  short                         dataRefIndex;
  long                          displayFlags;
  long                          textJustification;
  RGBColor                    bgColor;
  Rect                          defaultTextBox;
  ScrpSTElement                 defaultStyle;
  char                          defaultFontName[1];
};
```

The first five fields, of course, are the first five fields of the generic SampleDescription structure. The remaining fields are specific to text media. The displayFlags field holds a set of flags that indicate how the text is to be displayed. These flags allow us to specify various scrolling options and other positioning options. For the moment, we'll be content to specify the dfClipToTextBox flag, which restricts any updates caused by changes in the text track to the area occupied by the text track. (By all means, however, you should experiment with some of the scrolling options, like dfScrollIn and dfScrollOut.)

The defaultTextBox field specifies the location of the box that encloses the text. The rectangle is interpreted as relative to the upper-left corner of the text track rectangle. The textJustification field contains a value that specifies how the text is to be justified within the text box. The Movie Toolbox recognizes these constants for specifying a text justification (defined in the header file TextEdit.h):

```
enum {
  teFlushDefault          = 0,
  teCenter                = 1,
```

```
  teFlushRight              = -1,
  teFlushLeft               = -2
};
```

The bgColor field specifies the background color of the text box. The default text color is black. Note that because we call NewHandleClear to allocate a TextDescription structure, the default background color will also be black unless we change the values in the bgColor field. To make the text visible, we'll set the background color to white, like this:

```
RGBColor                myBGColor = {0xffff, 0xffff, 0xffff};

(**mySampleDesc).bgColor = myBGColor;
```

The last two fields of the TextDescription structure indicate the desired text style and font. We'll ignore these fields here.

Listing 5 shows a segment of the QTText_AddTextTrack function, which we use to add a new text track to a movie. As you can see, it allocates a handle to a text description structure, fills in some of the fields with appropriate values, calls PtrToHand and PtrAndHand to create the text media sample, and then calls AddMediaSample to add the text media sample to the text media.

Listing 5: Adding a text media sample

QTText_AddTextTrack

```
TextDescriptionHandle  mySampleDesc = NULL;
Handle                 mySample = NULL;
UInt16                 myLength;
RGBColor               myBGColor = {0xffff, 0xffff, 0xffff};

mySampleDesc = (TextDescriptionHandle)
                    NewHandleClear(sizeof(TextDescription));
if (mySampleDesc == NULL)
  goto bail;

(**mySampleDesc).descSize = sizeof(TextDescription);
(**mySampleDesc).dataFormat = TextMediaType;
(**mySampleDesc).displayFlags = dfClipToTextBox;
(**mySampleDesc).textJustification = teCenter;
(**mySampleDesc).defaultTextBox = myBounds;
(**mySampleDesc).bgColor = myBGColor;

myLength = EndianU16_NtoB(mySampleText[0]);

// create the text media sample: a 16-bit length word followed by the text
myErr = PtrToHand(&myLength, &mySample, sizeof(myLength));
if (myErr == noErr) {
  myErr = PtrAndHand((Ptr)(&mySampleText[1]), mySample,
                                        mySampleText[0]);
  if (myErr == noErr)
    AddMediaSample( myMedia, mySample, 0,
                    GetHandleSize(mySample),
                    myTextSampleDuration,
                    (SampleDescriptionHandle)mySampleDesc,
                    1, 0, NULL);
  DisposeHandle(mySample);
}

DisposeHandle((Handle)mySampleDesc);
```

The Movie Toolbox also provides the TextMediaAddTextSample function, which allows us to simplify this process significantly. Indeed, all of the work done in Listing 5 can be accomplished with this single line of code:

```
myErr = TextMediaAddTextSample(
                    myHandler,
                    (Ptr)(&mySampleText[1]),
                    mySampleText[0],
                    0,
                    0,
                    0,
                    NULL,
                    NULL,
```

```
    teCenter,
    &myBounds,
    dfClipToTextBox,
    0,
    0,
    0,
    NULL,
    myTextSampleDuration,
    NULL);
```

TextMediaAddTextSample takes seventeen parameters (count 'em!), which is probably some kind of record for a Movie Toolbox function. The payoff for this complexity is that it allows us to dispense with allocating a sample description or a text sample and with worrying about endian issues. Instead, we pass it the text media handler, myHandler (which we can obtain by calling GetMediaHandler on the text media), the text, and a handful of other parameters describing the desired characteristics of the text track.

Positioning a Text Track

When we create a text track, using either AddMediaSample or TextMediaAddTextSample, we need to specify the size and location of the text track. QTText determines the width of the new text track by calling GetTrackDimensions on the first video track in the movie:

```
GetTrackDimensions(myTypeTrack, &myWidth, &myHeight);
```

QTText uses the constant kTextTrackHeight (defined as 20 pixels) as the height of the text track.

For below-the-video text, we can specify the position of the text track by setting the track matrix, like this:

```
GetTrackMatrix(myTextTrack, &myMatrix);
TranslateMatrix(&myMatrix, 0, myHeight);
SetTrackMatrix(myTextTrack, &myMatrix);
```

All we've done here is translate the matrix downward by the height of the video track (myHeight). For text that overlays a video track, of course, we'll need to reset the matrix in some other way.

Enabling or Disabling a Text Track

Each track in a QuickTime movie is either *enabled* or *disabled*. By default, a newly-created track is enabled, in which case its media data directly contributes to the overall user experience. For example, an enabled video track is visible (unless of course it's completely covered by other enabled tracks), and an enabled audio track is audible. Most other media types, including text media, are visual media types, so once again being enabled means being visible. On the flip side, a disabled track does not usually contribute audible or visible data to the movie. Disabling a track is a quick and easy way to hide or mute it.

We can enable or disable a track by calling the SetTrackEnabled function, passing it a track identifier and a Boolean value that indicates whether to enable (true) or disable (false) the specified track. When we create a text track, we make sure it's visible by enabling it, like this:

```
SetTrackEnabled(myTextTrack, true);
```

We can hide the text track by passing false to disable it. Even if a text track is disabled, however, it can still be of use in a movie. For instance, we can search for text in a disabled text track, and the movie controller scans all text tracks, including disabled ones, when looking for chapter tracks. Similarly, the QuickTime Plug-In searches all text tracks, even disabled ones, when looking for an HREF track. Indeed, chapter tracks and HREF tracks are usually disabled.

Creating a Text Track

Listing 6 shows our complete function QTText_AddTextTrack for adding a text track to a movie. The parameter theStrings is an array of C strings; each element of that array is the text for a specific text sample. The parameter theFrames is an array of integers; each element of that array indicates how many video frames a text sample is to span. The sum of all the values in theFrames should equal the total number of frames in the video track. Finally, the isChapterTrack parameter indicates whether the new text track is to be a chapter track; if isChapterTrack is true, then the new text track is attached as a chapter track to the first track whose type is specified by the theType parameter.

Listing 6: Adding a text track

QTText_AddTextTrack

```
Track QTText_AddTextTrack (Movie theMovie,
  char *theStrings[], short theFrames[], short theNumFrames,
  OSType theType, Boolean isChapterTrack)
{
  Track             myTypeTrack = NULL;
  Track             myTextTrack = NULL;
  Media             myMedia = NULL;
  MediaHandler      myHandler = NULL;
  TimeScale         myTimeScale;
  MatrixRecord      myMatrix;
  Fixed             myWidth;
  Fixed             myHeight;
  OSErr             myErr = noErr;

  // get the (first) track of the specified type;
  // this track determines the width of the new text track
  // and (if isChapterTrack is true) is the target of the new chapter track
  myTypeTrack = GetMovieIndTrackType(theMovie, 1,
                theType, movieTrackMediaType);
  if (myTypeTrack == NULL)
    goto bail;

  // get the dimensions of the target track
  GetTrackDimensions(myTypeTrack, &myWidth, &myHeight);
  myTimeScale = GetMediaTimeScale
                (GetTrackMedia(myTypeTrack));

  // create the text track and media
  myTextTrack = NewMovieTrack(theMovie, myWidth,
                FixRatio(kTextTrackHeight, 1), kNoVolume);
  if (myTextTrack == NULL)
    goto bail;

  myMedia = NewTrackMedia(myTextTrack, TextMediaType,
                myTimeScale, NULL, 0);
  if (myMedia == NULL)
    goto bail;
```

```
myHandler = GetMediaHandler(myMedia);
if (myHandler == NULL)
  goto bail;

// figure out the text track geometry
GetTrackMatrix(myTextTrack, &myMatrix);
TranslateMatrix(&myMatrix, 0, myHeight);

SetTrackMatrix(myTextTrack, &myMatrix);
SetTrackEnabled(myTextTrack, true);

// edit the track media
myErr = BeginMediaEdits(myMedia);
if (myErr == noErr) {
  Rect            myBounds;
  short           myIndex;
  TimeValue       myTypeSampleDuration;
  TimeRecord      myTimeRec;

  myBounds.top = 0;
  myBounds.left = 0;
  myBounds.right = Fix2Long(myWidth);
  myBounds.bottom = Fix2Long(myHeight);

  // determine the duration of a sample in the track of the specified type
  myTypeSampleDuration =
              QTUtils_GetFrameDuration(myTypeTrack);


  for (myIndex = 0; myIndex < theNumFrames; myIndex++) {
    TimeValue     myTextSampleDuration;
    Str255        mySampleText;

    myTextSampleDuration = myTypeSampleDuration *
                           theFrames[myIndex];

    // set the time scale of the media to that of the movie
    myTimeRec.value.lo = myTextSampleDuration;
    myTimeRec.value.hi = 0;
    myTimeRec.scale = GetMovieTimeScale(theMovie);
    ConvertTimeScale(&myTimeRec,
                     GetMediaTimeScale(myMedia));
    myTextSampleDuration = myTimeRec.value.lo;

    QTText_CopyCStringToPascal(theStrings[myIndex],
                               mySampleText);

    // Listing 5 omitted at this point, for space reasons

    // write out the new data to the media
    myErr = TextMediaAddTextSample(
                    myHandler,
                    (Ptr)(&mySampleText[1]),
                    mySampleText[0],
                    0,
                    0,
                    0,
                    NULL,
                    NULL,
                    teCenter,
                      &myBounds,
                    dfClipToTextBox,
                    0,
                    0,
                    0,
                    NULL,
                    myTextSampleDuration,
                    NULL);
  }
}

myErr = EndMediaEdits(myMedia);
if (myErr != noErr)
  goto bail;

// insert the text media into the text track
myErr = InsertMediaIntoTrack(myTextTrack, 0, 0,
                    GetMediaDuration(myMedia), fixed1);
if (myErr != noErr)
  goto bail;

// set the text handling procedure
```

```
  TextMediaSetTextProc(myHandler, gTextProcUPP,

  (long)QTFrame_GetWindowObjectFromFrontWindow());

// if desired, set the new text track as a chapter track for the track of the specified type
  if (isChapterTrack)
    AddTrackReference(myTypeTrack, myTextTrack,
                      kTrackReferenceChapterList, NULL);

bail:
  return(myTextTrack);
}
```

For the moment, you can ignore the calls to TextMediaSetTextProc and AddTrackReference. We'll explain them a little later.

Text Searching

The Movie Toolbox provides several functions that we can use to search for a specific word or series of words in a text track. If we are interested in simply finding out where in a text track the next occurrence of a string is located, we can use the TextMediaFindNextText function, like this:

```
myTimeValue = GetMovieTime(myMovie, NULL);
myErr = TextMediaFindNextText(myHandler,
                              (Ptr)(&theText[1]),
                              theText[0],
                              myFlags,
                              myTimeValue,
                              &myFoundTime,
                              &myFoundDuration,
                              &gOffset);
```

The first parameter, myHandler, is the text media handler associated with the text track. The second and third parameters specify the text to be searched for and the length of that text; here we're supposing that the text is contained in the variable theText, which is a Pascal string. The fourth parameter is a set of *search flags*, which indicate how TextMediaFindNextText is to search for the specified text. These flags are defined:

```
enum {
  findTextEdgeOK                  = 1 << 0,
  findTextCaseSensitive             = 1 << 1,
  findTextReverseSearch             = 1 << 2,
  findTextWrapAround              = 1 << 3,
  findTextUseOffset                 = 1 << 4
};
```

These constants are pretty much self-explanatory, except for the first and the last. If findTextEdgeOK is set in the search flags, then TextMediaFindNextText will match text beginning at the movie time specified by the fifth parameter; otherwise, the text must occur in some later (or earlier, if findTextReverseSearch is set) sample. If findTextUseOffset is set, then TextMediaFindNextText will search beginning at the offset specified by the last parameter. This allows us to find separate occurrences of the search text in a single text sample.

Our QTText sample application maintains a couple of global variables that keep track of the kind of search the user wants to perform. We'll use those variables to set our search flags like this:

```
myFlags = findTextUseOffset;
if (!gSearchForward)
  myFlags |= findTextReverseSearch;
if (gSearchWrap)
  myFlags |= findTextWrapAround;
if (gSearchWithCase)
  myFlags |= findTextCaseSensitive;
```

If TextMediaFindNextText finds the text specified by the second and third parameters in some text sample, it returns the movie time of the beginning of that sample in the sixth parameter (here, &myFoundTime). It also returns the duration of that sample in the seventh parameter and, in the last parameter, the byte offset (from the beginning of the text portion of that sample) of the first character of that text.

Typically, we don't just want to find out where some text begins; we also want to advance the movie to that point and highlight the found text. We can use the MCDoAction function with the mcActionGoToTime action to set the current movie time to the time returned to us by TextMediaFindNextText, like so:

```
myNewTime.value.hi = 0;
myNewTime.value.lo = myFoundTime;
myNewTime.scale = GetMovieTimeScale(myMovie);
myNewTime.base = NULL;

// go to the found text
MCDoAction(myMC, mcActionGoToTime, &myNewTime);
```

And we can use the TextMediaHiliteTextSample function to highlight the selected text:

```
myColor.red = myColor.green = myColor.blue = 0x8000;  // grey
TextMediaHiliteTextSample(myHandler, myFoundTime, gOffset,
  gOffset + theText[0], &myColor);
```

Once again, however, the Movie Toolbox provides a function that greatly simplifies our work here. The MovieSearchText function, introduced in QuickTime version 2.0, finds the text, sets the movie time to the beginning of the text sample containing that text, and highlights the found text in that sample. So we can replace all the code we've encountered so far in this section with this single line of code:

```
myErr = MovieSearchText(myMovie,
                        (Ptr)(&theText[1]),
                        theText[0],
                        myFlags,
                        NULL,
                        &myTimeValue,
                        &gOffset);
```

When we call MovieSearchText, we pass in the movie to search, the search text and search text length, a set of flags, the first text track to search, and the movie time at which to start the search. The set of flags can include any of the search flags listed above, as well as any of these additional flags that are specific to the MovieSearchText function:

```
enum {
  searchTextDontGoToFoundTime       = 1L << 16,
```

// Like most Mac developers,

// I easily spend 12 hours a day

// staring at line after line of C++ code in tiny, 9 point Monaco.
// Sometimes it makes my eyes feel like they're on fire.

{

So the last thing I need is some
fuzzy monitor that adds to my headaches.

}

/****** begin excitement ******/

// That's why I'm so jazzed about this SGI monitor.

// Its ultra-high resolution = 1600 x 1024 and dpi = 110,
// giving me razor-sharp contrast.
// And the high refresh rate is
// perfect for poring through lines of code.

// At first, I was amazed at the clarity, the fine details that emerged.

{

It was like seeing things for the first time.

}

// Later, though, I learned to appreciate the wide aspect ratio [= 16:10],
// with a generous 17.3 inches of viewing area. SGI's 1600SW
// lets me have all my documents viewable at once, and it's
// a flat panel so it fits on my desk with room to spare.

// From the moment I saw this thing I was hooked. You will be, too.

{

Especially when you find out
how affordable it is.

}

// Check it out.

/****** end excitement ******/

Michael Whittingham
V.P., Engineering
Wired, Inc.

www.sgi.com/flatpanel

This advertisement was written by an actual engineer. This is the first time in 7 months that he's been seen during the day.

sgi™

```
  searchTextDontHiliteFoundText      = 1L << 17,
  searchTextOneTrackOnly             = 1L << 18,
  searchTextEnabledTracksOnly        = 1L << 19
};
```

Including either of the first two flags, searchTextDontGoToFoundTime and searchTextDontHiliteFoundText, allows us to override the default "go-to-and-highlight" behavior of MovieSearchText. The next two flags modify the track-searching behavior. If we pass a track identifier in the fifth parameter, then MovieSearchText will search only that track if the searchTextOneTrackOnly flag is set; otherwise, it will search all text tracks in the specified movie, starting with that track. We can restrict the search to all enabled text tracks by setting the searchTextEnabledTracksOnly flag.

If MovieSearchText finds the specified text, it returns the movie time of the text sample in which the text was found and the byte offset within that sample of the found text. It also returns the track identifier of the track containing the text sample, unless the track parameter was set to NULL on input.

Listing 7 contains the definition of our function QTText_FindText, which we use to search for text. As you can see, it uses either the TextMediaFindNextText function or the MovieSearchText function, depending on the value of the compiler flag USE_MOVIESEARCHTEXT.

Listing 7: Finding some text

QTText_FindText

```
void QTText_FindText (WindowObject theWindowObject,
                                          Str255 theText)
{
  ApplicationDataHdl      myAppData = NULL;
  Movie                   myMovie = NULL;
  MediaHandler            myHandler = NULL;
  MovieController         myMC = NULL;
  long                    myFlags = 0L;
  TimeValue               myTimeValue;
  OSErr                   myErr = noErr;

  myAppData = (ApplicationDataHdl)
        QTFrame_GetAppDataFromWindowObject(theWindowObject);
  if (myAppData == NULL)
    return;

  myMC = (**theWindowObject).fController;
  myMovie = (**theWindowObject).fMovie;
  myHandler = (**myAppData).fTextHandler;

  // set the search features
  myFlags = findTextUseOffset;
  if (!gSearchForward)
    myFlags |= findTextReverseSearch;
  if (gSearchWrap)
    myFlags |= findTextWrapAround;
  if (gSearchWithCase)
    myFlags |= findTextCaseSensitive;

  myTimeValue = GetMovieTime(myMovie, NULL);

#if USE_MOVIESEARCHTEXT
  myFlags |= searchTextEnabledTracksOnly;

  myErr = MovieSearchText(myMovie, (Ptr)(&theText[1]),
        theText[0], myFlags, NULL, &myTimeValue, &gOffset);
  if (myErr != noErr)
    QTFrame_Beep();     // if the desired string wasn't found, beep
#else
  if (myHandler != NULL) {
    TimeValue  myFoundTime, myFoundDuration;
    TimeRecord myNewTime;
    RGBColor myColor;

  myColor.red = myColor.green = myColor.blue = 0x8000;// grey

    // search for the specified text
    myErr = TextMediaFindNextText(myHandler,
      (Ptr)(&theText[1]), theText[0], myFlags, myTimeValue,
        &myFoundTime, &myFoundDuration, &gOffset);
    if (myFoundTime != -1) {
      // convert the TimeValue to a TimeRecord
      myNewTime.value.hi = 0;
      myNewTime.value.lo = myFoundTime;
      myNewTime.scale = GetMovieTimeScale(myMovie);
      myNewTime.base = NULL;

      // go to the found text
      MCDoAction(myMC, mcActionGoToTime, &myNewTime);

      // highlight the text
      TextMediaHiliteTextSample(myHandler, myFoundTime,
                gOffset, gOffset + theText[0], &myColor);

    } else {
      QTFrame_Beep();     // if the desired string wasn't found, beep
    }
  }
#endif

  // update the current offset, if we're searching forward
  if (gSearchForward && (myErr == noErr))
    gOffset += theText[0];
}
```

Of course, *your* code won't need to use this compiler flag; you'll call just MovieSearchText or TextMediaFindNextText for your text searching. Here we simply want to illustrate how to call both of these functions.

Text Editing

Let's consider now how to edit the data in a text track. Conceptually, this is a fairly simple operation. We can just call DeleteTrackSegment to delete one or more existing text samples from a track; then we can call TextMediaAddTextSample to add a new text sample to the text media and then InsertMediaIntoTrack to place that text sample at the desired location in the track. For the moment, we'll limit ourselves to replacing a single existing text sample by another sample that occupies the same location in the track (that is, that has the same starting point and duration as the original sample).

When the user selects the "Edit Current Text..." menu item, we'll display the dialog box shown in **Figure 12**. If the user clicks the OK button, we'll retrieve the text from the edit text control in that dialog box and use that text as the replacement text data. There are only two things we still need to figure out: (1) how can we get the text of the current text sample (to put into the dialog box when it's first displayed)? And, (2) how can we determine the starting time and duration of the current text sample? Let's take these tasks in order.

Some companies are really on the ball.

POWERED BY webex

Some of the world's most forward-thinking companies have transformed their Web sites into live meeting rooms. They have discovered the speed, reliability and scalability of the WebEx Interactive Network. Their Web sites now hum with people conducting business right in their browsers, without any hardware or software changes. Now you can give presentations. Share software and desktops. Tour the Web. Voice and video conference. All in real-time. All on the Web. Talk about ROI. Isn't it time your company got on the ball? Visit webex.com or call 1-877-50-WebEx to see how great minds meet online.

great minds meet online at webex.com webex™

Dept. #4020

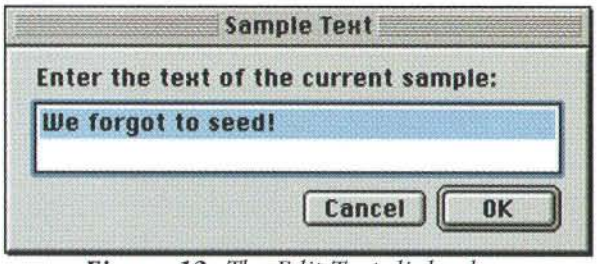


Figure 12: *The Edit Text dialog box.*

Getting the Current Text

The first task is the easier of the two, mainly because whenever the text media handler is about to display a new text sample, it calls an application-defined *text callback procedure* that we've previously installed by calling TextMediaSetTextProc (in Listing 6). The text callback procedure is passed several parameters, one of which is a handle to the sample data of the current media sample. So all we need to do is make a copy of the sample text in a place we can find it when we are about to display the dialog box shown in Figure 12. Listing 8 shows our application's text callback procedure.

Listing 8: Getting the current text

QTText_TextProc

```
PASCAL_RTN OSErr QTText_TextProc (Handle theText,
    Movie theMovie, short *theDisplayFlag, long theRefCon)
{
#pragma unused(theMovie, theRefCon)
  char        *myTextPtr = NULL;
  short       myTextSize;
  short       myIndex;

  // on entry to this function, theText is a handle to the text sample data,
  // which is a big-endian 16-bit length word followed by the text itself
  myTextSize = EndianU16_BtoN(*(short *)(*theText));
  myTextPtr = (char *)(*theText + sizeof(short));

  // copy the text into our global variable
  for (myIndex = 1; myIndex <= myTextSize;
                              myIndex++, myTextPtr++)
    gSampleText[myIndex] = *myTextPtr;

  gSampleText[0] = myTextSize;

  // ask for the default text display
  *theDisplayFlag = txtProcDefaultDisplay;

  return(noErr);
}
```

As you can see, we first parse the sample data to get the 16-bit length field and the location of the first character in the text string. Then we copy the characters into the global variable gSampleText, which is of type Str255. Finally, we return the value txtProcDefaultDisplay in the parameter theDisplayFlag; this instructs the text media handler to use the display flags contained in the displayFlags field of the text description structure for that text sample. (There are also constants to force the sample to be shown or not shown, regardless of the media's default display flags.)

Finding Sample Boundaries

Now we need to figure out how to find the starting time and duration of the current text media sample (so we know what segment of the text track to replace). An easy way to get the starting time would be to call GetMovieTime in our text callback procedure and then assign the returned value to a global variable. A better way — because it can be used with media types other than text — is to call the GetTrackNextInterestingTime function inside the QTText_EditText function. GetTrackNextInterestingTime allows us to search for specific times in a track, given a set of search criteria. The search criteria are specified by these flags:

```
enum {
  nextTimeMediaSample              = 1 << 0,
  nextTimeMediaEdit                = 1 << 1,
  nextTimeTrackEdit                = 1 << 2,
  nextTimeSyncSample             = 1 << 3,
  nextTimeStep                     = 1 << 4,
  nextTimeEdgeOK                   = 1 << 14,
  nextTimeIgnoreActiveSegment      = 1 << 15
};
```

For present purposes, we'll use the two flags nextTimeMediaSample and nextTimeEdgeOK, which tell GetTrackNextInterestingTime to search in the next sample in the track's media but to consider samples that begin or end at the search starting time.

We'll begin by getting the current movie time (which might not be the beginning of the current text sample), like this:

```
myMovieTime = GetMovieTime(myMovie, NULL);
```

Then we want to search *backward* to find the beginning of the current media sample:

```
GetTrackNextInterestingTime(
                    myTrack,
                    nextTimeEdgeOK | nextTimeMediaSample,
                    myMovieTime,
                    -fixed1,
                    &myInterestingTime,
                    NULL);
```

The third parameter specifies the starting time for the search and the fourth parameter indicates the direction of the search; because the value here is negative, the search goes backwards from the current movie time. Once GetTrackNextInterestingTime finds the beginning of the current media sample, it returns that time in the location pointed to by the fifth parameter. We've set the sixth parameter to NULL because we don't need the duration from the current time to the found interesting time to be returned to us.

So we've found the beginning of the current text sample. We can find the duration of that sample by calling

Works with Apple AirPort

SkyLINE™ 11Mb

Join the wireless revolution.

802.11b for Mac OS and Windows

Wireless LAN access to email, Internet, printers & more!

Farallon®

www.farallon.com

Download Free Wireless White Paper Today!

or Call (800) 613-4954

Available @ the DEPOT™

GetTrackNextInterestingTime once more, this time searching forward from the beginning of the sample, like this:

```
myMovieTime = myInterestingTime;
GetTrackNextInterestingTime(
                        myTrack,
                        nextTimeEdgeOK | nextTimeMediaSample,
                        myMovieTime,
                        fixed1,
                        NULL,
                        &myDuration);
```

In this case, we want only the duration of the sample returned to us, so we pass NULL in the fifth parameter and &myDuration in the sixth.

Keep in mind that the time values that GetTrackNextInterestingTime returns to us are in the movie time scale. This is useful, since the parameters to DeleteTrackSegment must also be in the movie time scale. So we can now call DeleteTrackSegment to remove the current text sample from the track:

```
myErr = DeleteTrackSegment(myTrack, myInterestingTime,
                                    myDuration);
```

All that remains is to add a new text sample in place of the one we just removed. For this, we can call TextMediaAddTextSample as we did earlier. There is only one complication here: the duration we pass to TextMediaAddTextSample must be expressed in the media time scale, not the movie time scale. But the Movie Toolbox conveniently provides the MediaTimeToSampleNum function that we can use to get the start time and duration of the current media sample in the media time scale, like this:

```
myMovieTime = GetMovieTime(myMovie, NULL);
myMediaCurrentTime = TrackTimeToMediaTime
                            (myMovieTime, myTrack);
MediaTimeToSampleNum(
                    myMedia,
                    myMediaCurrentTime,
                    &myMediaSampleIndex,
                    &myMediaSampleStartTime,
                    &myMediaSampleDuration);
```

So, we've got all the information we need to call TextMediaAddTextSample and InsertMediaIntoTrack and thereby complete the text sample editing operation. For the complete definition of QTText_EditText, see the file QTText.c.

Chapter Tracks

We learned earlier that a chapter track is just a text track that has been associated in a particular way with some other track. Let's call this other track the *target track*. We create the association between a text track and the target track by creating a *track reference* from that target to the text track. In general, a track reference is simply a way for one track to establish a relationship with some other track. The type of the track reference indicates the nature of that relationship. The Movie Toolbox currently provides three constants for track reference types:

```
enum {
  kTrackReferenceChapterList    = FOUR_CHAR_CODE('chap'),
  kTrackReferenceTimeCode        = FOUR_CHAR_CODE('tmcd'),
  kTrackReferenceModifier        = FOUR_CHAR_CODE('ssrc')
};
```

A track reference of type kTrackReferenceChapterList is used to create a chapter track. A track reference of type kTrackReferenceTimeCode is used to create a *timecode track*, in which timecode values are associated with the samples of the target track. (We'll consider timecode tracks in more detail in the next *QuickTime Toolkit* article.) A track reference of type kTrackReferenceModifier is used to create a *modifier track*; modifier tracks are useful when you want one track to modify the appearance or behavior of a target track. For instance, a tween track is a kind of modifier track that can be used to change, say, the volume of a sound track as the movie progresses. We'll encounter modifier tracks in several upcoming articles, to change the current image of a sprite track and to apply a special effect to a video track. Other parts of QuickTime define additional types of track references. For example, the file QuickTimeVRFormat.h defines several types of track references that are used in building QuickTime VR movie files.

It's actually a rather trivial operation to create a chapter track once we have a text track at hand. If myTypeTrack is a track identifier for a target track (in the present case, a video track), then we can create a chapter track reference to our text track like this:

```
AddTrackReference(myTypeTrack, myTextTrack,
                    kTrackReferenceChapterList, NULL);
```

The last parameter is a pointer to a long word in which AddTrackReference will return the index assigned to the new track reference; we don't need this information, so we set that parameter to NULL.

All the chapter titles must be contained in a single text track; we specify the starting time for chapters when we add the text to the text track by calling TextMediaAddTextSample. Note that we need to create the chapter association only between the text track and one other target track, not between the text track and all other tracks in the movie. The target track must be enabled, but typically the chapter track is not enabled (unless we want the text track to be visible).

It's also very easy to remove a track reference and hence to change a chapter track back into a non-chapter text track. Again, if myTypeTrack is the target track, then we can disassociate it from the text track by calling DeleteTrackReference, like this:

```
DeleteTrackReference(myTypeTrack,
                    kTrackReferenceChapterList, 1);
```

It just takes one bug...

The sooner you find that last bug,
the sooner you can ship,
and the sooner you can sleep.

Find these and other essential developer tools at Developer DEPOT®

Spotlight 1.0

only $189!

"Automatic Debugging" on the Macintosh. Easy to use, Spotlight can automatically find run time bugs in your code without your having to change one line of your code. Nail down random bugs immediately. Ever dereference the wrong pointer? Ever pass the wrong value to a toolbox command or over write an array? Your compiler often lets the code compile fine and you may not find that bug until you've burned it into CD-ROM or released it to the net. Find the bugs now, kill them easily, get to sleep sooner.

BugLink Solo

only $79!

The only thing more frustrating that bugs, is bug reports. Users will send in everything from war and peace to "it crashed." BugLink Solo let's you define what information you want in a bug report, then include a customized BugReporter application with your application. Users simply click open the reporter, fill in the boxes, and click send. The report is transparently sent to the email address you defined. BugLink can login into that eMail account, download each report, and present you with an organized, complete, description of each bug. Great for shareware developers, system administrators, and web developers!

DCon 1.0

only $39.95!

Every doctor knows, the secret is listening to the patient. The most frustrating part about debugging on Macintosh is you can't "printf" to show the state of a variable or position in your code. DCon let's your code to talk to you. This system extension adds a console window and file logging services to Macintosh and can be used to record and display status and debugging information during development. It can be called from virtually any code, any where at any time – even from interrupt handlers, I/O completion routines, VBL tasks, deferred tasks, and more! DCon does not allocate memory in any Mac developers debugging arsenal.

www.devdepot.com

PO Box 5200 Westlake Village, CA 91359-5200 • Voice: 800/MACDEV-1 (800/622-3381)
Outside US/Canada: 805/494-9797 • Fax: 805/494-9798 • Email: orders@devdepot.com

Here the last parameter is the index of the track reference of the specified type that we want to remove. Our code attaches at most one chapter track to a target, so we can safely set that index to 1.

Listing 9 shows our complete function for turning a text track into a chapter track or turning a chapter track back into a text track. The Boolean parameter isChapterTrack determines whether the first text track in the movie becomes a chapter track or is demoted from that lofty rank.

Listing 9: Setting and unsetting chapter tracks

QTText_SetTextTrackAsChapterTrack

```
OSErr QTText_SetTextTrackAsChapterTrack
    (WindowObject theWindowObject, OSType theType,
                                Boolean isChapterTrack)
{
  ApplicationDataHdl    myAppData = NULL;
  Movie                 myMovie = NULL;
  MovieController       myMC = NULL;
  Track                 myTypeTrack = NULL;
  Track                 myTextTrack = NULL;
  OSErr                 myErr = paramErr;

  // get the movie, controller, and related stuff
  myAppData = (ApplicationDataHdl)
        QTFrame_GetAppDataFromWindowObject(theWindowObject);
  if (myAppData == NULL)
    return(myErr);

  myMovie = (**theWindowObject).fMovie;
  myMC = (**theWindowObject).fController;
  myTextTrack = (**myAppData).fTextTrack;

  if ((myMovie != NULL) && (myMC != NULL)) {
    myTypeTrack = GetMovieIndTrackType(myMovie, 1, theType,
              movieTrackMediaType | movieTrackEnabledOnly);
    if ((myTypeTrack != NULL) && (myTextTrack != NULL)) {
      // add or delete a track reference, as determined by the desired final state
      if (isChapterTrack)
        myErr = AddTrackReference(myTypeTrack, myTextTrack,
                        kTrackReferenceChapterList, NULL);
      else
        myErr = DeleteTrackReference(myTypeTrack,
                        kTrackReferenceChapterList, 1);

      // tell the movie controller we've changed aspects of the movie
      MCMovieChanged(myMC, myMovie);

      // stamp the movie as dirty
      (**theWindowObject).fIsDirty = true;
    }
  }

  return(myErr);
}
```

Note that after we call AddTrackReference or DeleteTrackReference, we need to call MCMovieChanged to inform the movie controller that we've changed the associated movie. This prompts the movie controller to redraw the movie controller bar to show or hide the chapter pop-up menu.

The file QTText.c contains a number of other chapter track utilities. Listing 10 defines the one we use for determining whether a track is a chapter track; we call this function when we need to determine whether to place a check mark next to the "Chapter Track" menu item.

Listing 10: Determining whether a track is a chapter track

QTText_IsChapterTrack

```
Boolean QTText_IsChapterTrack (Track theTrack)
{
  Movie         myMovie = NULL;
  Track         myTrack = NULL;
  long          myTrackCount = 0L;
  long          myTrRefCount = 0L;
  long          myTrackIndex;
  long          myTrRefIndex;

  myMovie = GetTrackMovie(theTrack);
  if (myMovie == NULL)
    return(false);

  myTrackCount = GetMovieTrackCount(myMovie);
  for (myTrackIndex = 1; myTrackIndex <= myTrackCount;
                                        myTrackIndex++) {
    myTrack = GetMovieIndTrack(myMovie, myTrackIndex);
    if ((myTrack != NULL) && (myTrack != theTrack)) {

      // iterate thru all track references of type kTrackReferenceChapterList
      myTrRefCount = GetTrackReferenceCount(myTrack,
                              kTrackReferenceChapterList);
      for (myTrRefIndex = 1; myTrRefIndex <= myTrRefCount;
                                        myTrRefIndex++) {
        Track  myRefTrack = NULL;

        myRefTrack = GetTrackReference(myTrack,
                kTrackReferenceChapterList, myTrRefIndex);
        if (myRefTrack == theTrack)
          return(true);
      }
    }
  }

  return(false);
}
```

Hypertext Reference Tracks

A hypertext reference track, or HREF track, is a text track in which some or all of the samples contain hypertext links, in the form of URLs. (Actually, there's no requirement that any of the samples in an HREF track contain a hypertext link, but then of course it's not very useful.) These URLs can be any kind of URL supported by QuickTime, including HTTP, HTTPS, FTP, file, RTSP, and JavaScript URLs. Indeed, if the QuickTime Plug-In finds a URL it doesn't recognize, it passes it to the web browser for processing. So, really, the sky's the limit in terms of the kind of URLs we can put in an HREF track.

From a programming perspective, creating an HREF track is even easier than creating a chapter track. All we need to do is set the name of a text track to "HREFTrack". The plug-in interprets the first text track in a movie having that name as the active HREF track. Listing 11 defines the function QTText_SetTextTrackAsHREFTrack that we can use to set and unset a text track as an HREF track.

Listing 11: Setting and unsetting HREF tracks

QTText_SetTextTrackAsHREFTrack

```
OSErr QTText_SetTextTrackAsHREFTrack
                    (Track theTrack, Boolean isHREFTrack)
{
  OSErr    myErr = noErr;

  myErr = QTUtils_SetTrackName(theTrack,
        isHREFTrack ? kHREFTrackName : kNonHREFTrackName);

  return(myErr);
}
```

A track's name is stored as part of the track's user data, so QTUtils_SetTrackName (defined in QTUtilities.c) calls SetUserDataItem to set the name. In QTText_SetTextTrackAsHREFTrack, we use these constants for the track names:

```
#define kHREFTrackName      "HREFTrack"
#define kNonHREFTrackName   "Text Track"
```

Ideally, each track should have a unique name (though this is not required). So instead of hard-coding the name for the non-HREF track, we can generate a track name dynamically, looking at the names that are already assigned to tracks in the movie. QTUtilities.c defines a function, QTUtils_MakeTrackNameByType, that we can call to accomplish this. Reworking QTText_SetTextTrackAsHREFTrack to use QTUtils_MakeTrackNameByType is left as an exercise for the reader.

Occasionally it's useful to know whether a specified text track is an HREF track. (For instance, QTText needs to know this to decide whether to put a check mark beside the "HREF Track" menu item.) The function QTText_IsHREFTrack, defined in Listing 12, returns a Boolean value that indicates whether a given text track is an HREF track.

Listing 12: Determining whether a track is an HREF track

QTText_IsHREFTrack

```
Boolean QTText_IsHREFTrack (Track theTrack)
{
  Boolean    isHREFTrack = false;
  char       *myTrackName = NULL;

  myTrackName = QTUtils_GetTrackName(theTrack);
  if (myTrackName != NULL)
  isHREFTrack = (strcmp(myTrackName, kHREFTrackName) == 0);

  free(myTrackName);
  return(isHREFTrack);
}
```

Some Loose Ends

Let's finish up by taking care of a few loose ends in our basic application framework that become apparent when we start working with text tracks. As

you know, when we paste some data into a movie, the current movie time is set to the time immediately following the pasted data. (This is the standard behavior with any kind of pasting.) If pasting causes the movie box to expand, it might happen that the expanded portion of the movie box in the current frame contains areas that should be erased but which are not erased by the movie controller. For example, if we've added some text in parallel, we might see something like **Figure 13**. Here, the video media handler has redrawn the video portion of the movie box but the text media handler, thinking (correctly) that there's no text for the current movie time, has left the text portion of the movie box untouched. As a result, the image of the movie controller bar, which used to occupy the space now occupied by the text track, is not erased. This is not good, but it's easy enough to fix.

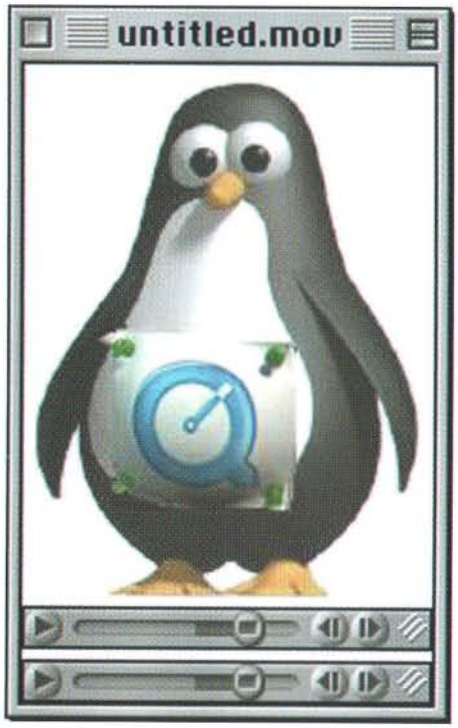


Figure 13: *A movie window after adding in parallel.*

When some part of the movie box needs to be redrawn, an update event is generated for that portion of the movie box. When we pass that event to MCIsPlayerEvent, the movie controller redraws the appropriate portion of the movie and clears that area from the update region of the window. The problem, as we've just seen, is that the movie controller doesn't think that the bottom portion needs to be redrawn and hence doesn't redraw it. We can solve this problem by erasing that portion of the window ourselves. Listing 13 shows our updated version of QTApp_Draw.

Listing 13: Redrawing a movie window

QTApp_Draw

```
void QTApp_Draw (WindowReference theWindow)
{
  GrafPtr      mySavedPort = NULL;
  GrafPtr      myWindowPort = NULL;
  WindowPtr    myWindow = NULL;
  Rect           myRect;

  GetPort(&mySavedPort);
  myWindowPort =
          QTFrame_GetPortFromWindowReference(theWindow);
myWindow = QTFrame_GetWindowFromWindowReference(theWindow);

  if (myWindowPort == NULL)
    return;

  MacSetPort(myWindowPort);

#if TARGET_API_MAC_CARBON
  GetPortBounds(myWindowPort, &myRect);
#else
  myRect = myWindowPort->portRect;
#endif

  BeginUpdate(myWindow);

  if (QTFrame_IsDocWindow(theWindow))
    EraseRect(&myRect);

  // ***insert application-specific drawing here***

  EndUpdate(myWindow);
  MacSetPort(mySavedPort);
}
```

As you can see, we call EraseRect on the entire window rectangle. Keep in mind, however, that BeginUpdate limits the redrawn portion to the intersection of the visible region of the window and the current update region. Since MCIsPlayerEvent will already have removed the active movie region from the update region, our call to EraseRect just redraws the visible portion of the update region that wasn't redrawn by the movie controller.

The last thing we need to do is make sure that the entire movie box is included in the update region when we call MCIsPlayerEvent. We can accomplish this by adding a few lines to our QTFrame_HandleEditMenuItem function. Essentially, we need to make sure to invalidate the entire movie box whenever the size of the movie box might have changed. Listing 14 shows the lines we'll add to the end of QTFrame_HandleEditMenuItem.

Listing 14: Invalidating a movie window

QTFrame_HandleEditMenuItem

```
  // if the size of the movie might have changed, invalidate the entire movie
box
  if ((theMenuItem == IDM_EDITUNDO) ||
      (theMenuItem == IDM_EDITCUT) ||
      (theMenuItem == IDM_EDITPASTE) ||
      (theMenuItem == IDM_EDITCLEAR)) {
    Rect     myRect;
#if TARGET_OS_WIN32
    RECT     myWinRect;
#endif
```

```
    MCGetControllerBoundsRect(myMC, &myRect);
#if TARGET_OS_MAC
    InvalWindowRect(QTFrame_GetWindowFromWindowReference
                        (theWindow), &myRect);
#endif
#if TARGET_OS_WIN32
    QTFrame_ConvertMacToWinRect(&myRect, &myWinRect);
    InvalidateRect(theWindow, &myWinRect, false);
#endif
  }
```

With these changes made, we should see no glitches like those in **Figure 13**.

Conclusion

We've covered a fair amount of ground in this article. We've seen how to create text tracks, chapter tracks, and HREF tracks; we've also learned how to search a text track and edit the data in a text track. Along the way, we've seen how to upgrade our Edit menu item adjusting and our movie window redrawing, so that even our applications that are not directly concerned with text can import text from files or from the system scrap.

We've also learned a more general lesson: QuickTime often provides more than one way to accomplish some particular task. We've seen, for instance, that we can call either AddMediaSample or TextMediaAddTextSample to add media samples to a text track. And, we can call either TextMediaFindNextText or MovieSearchText to search for text within a text track. Which of these functions we use in any particular case is a matter of taste, no doubt, but also a matter of simplicity and code size. TextMediaAddTextSample and MovieSearchText hold the clear advantage when we consider the amount of source code we need to write and the kinds of details (like endian issues) that we need to attend to. In the future, we'll generally opt for the simpler, cleaner way of solving our programming tasks (and leave the dinosaur bones for the archeologists).

Acknowledgements and References

Thanks are due once again to Brian Friedkin, for his ever-helpful guidance on Windows-related issues. Some of the code in QTText is based on an earlier sample code package by Nick Thompson; you can find an explanation of that code in his article in *develop*, Issue 20 (archived at http://www.mactech.com/articles/develop/issue_20/20quicktime.html).

You can find a thorough explanation of text descriptors at http://www.apple.com/quicktime/authoring/textdescriptors.html; an even more readable account is found in Steve Gulie's indispensable book *QuickTime for the Web* (available at http://www.devdepot.com/ and at all good bookstores).

by Jeff Clites <online@mactech.com>

A few months ago we covered Quartz, Mac OS X's new 2D graphics technology based on the imaging model of Adobe's PDF. Quartz will be responsible, directly or indirectly, for most of what the user sees on the screen. What we didn't emphasize directly is that Quartz is 2D-only. Under Mac OS X, 3D graphics will be based on OpenGL. So what's OpenGL?

OpenGL is in, QuickDraw 3D is out

OpenGL is a 3D graphics library developed by SGI (then called Silicon Graphics). It's a high-performance, cross-platform library, available on Mac OS 9, Mac OS X, Windows, and many Unix variants, and many video cards provide hardware acceleration. It is the library of choice for sophisticated 3D development, most notably the gaming industry. In particular, OpenGL is the basis if id software's Quake games, and according to the company allows them to develop their products with only a tiny amount of platform-specific code. This is important for the Macintosh community, because it makes it that much more likely that games and other 3D-graphics-intensive software (such as scientific data visualization, CAD, and architectural design packages) will be available.

If you've done any 3D programming on the Macintosh in the past, you probably used QuickDraw 3D, the 3D library on Mac OS 9 and, technically, part of QuickTime (and therefore available on Windows as well). The sad news is that OpenGL is replacing QuickDraw 3D on Mac OS X, not just providing an alternative. The worst part is that, while OpenGL is the industry standard, QD-3D is much easier to use for simple 3D graphics. In fact, you could make the argument that OpenGL has a prohibitive learning curve for anyone wanting to do something simple—for instance, creating a 3D bar graph. It has a large procedural API, and it's yet another conceptual barrier you have to cross to get your application written. It also lacks QD-3D's elegant object-oriented (though still C-based) API, and well as its file format for storing 3D objects. On the other hand, you could make a strong counterargument that "simple" uses of 3D graphics are few and far between, and somewhat of an oxymoron—when's the last time you saw 3D graphics outside of a game, a screen saver, or a splash screen? In this light, there are certainly more people cheering the availability of OpenGL than are mourning the loss of QD-3D, but it would have been nice to retain an easy-to-use, high-level API, possibly layered on top of OpenGL. Although they seem to fit well together, there's no indication that Apple will go this route, and if there really are very few developers using QD-3D, it makes sense not to devote the resources to it.

Enter Quesa

Now the good news. The good news is Quesa, a third-party, open-source (LGPL) effort to recreate QD-3D from scratch—in other words, and independent library which is API-compatible with QD-3D. This approach is really a win-win situation for the Macintosh community, because it frees Apple from the burden of maintaining an API which may not be widely used, yet it will still be available to those who need it, for as long as anyone is interested enough in it to maintain it. The needs-based focus of many open-source projects is clearly present in Quesa, which was started because its founder wanted to be able to run his 3D screen saver on Mac OS X. The project is under active development, and in fact is mostly complete at this point. As a side benefit of the open-source approach, it is now truly cross-platform, available now on Mac OS 9, Mac OS X, Windows, and Linux, and coming to the BeOS as well. And as you might expect, Quesa can run on top of OpenGL, and so it will benefit from OpenGL hardware acceleration. As mentioned above, QD-3D (and hence Quesa) is object-oriented although written in C. This is interesting from a design standpoint, and is similar in spirit to Apple's new CoreFoundation API, which gives C-based access to key data types and APIs which originated in Cocoa. You can read more about Quesa's structure on its documentation page.

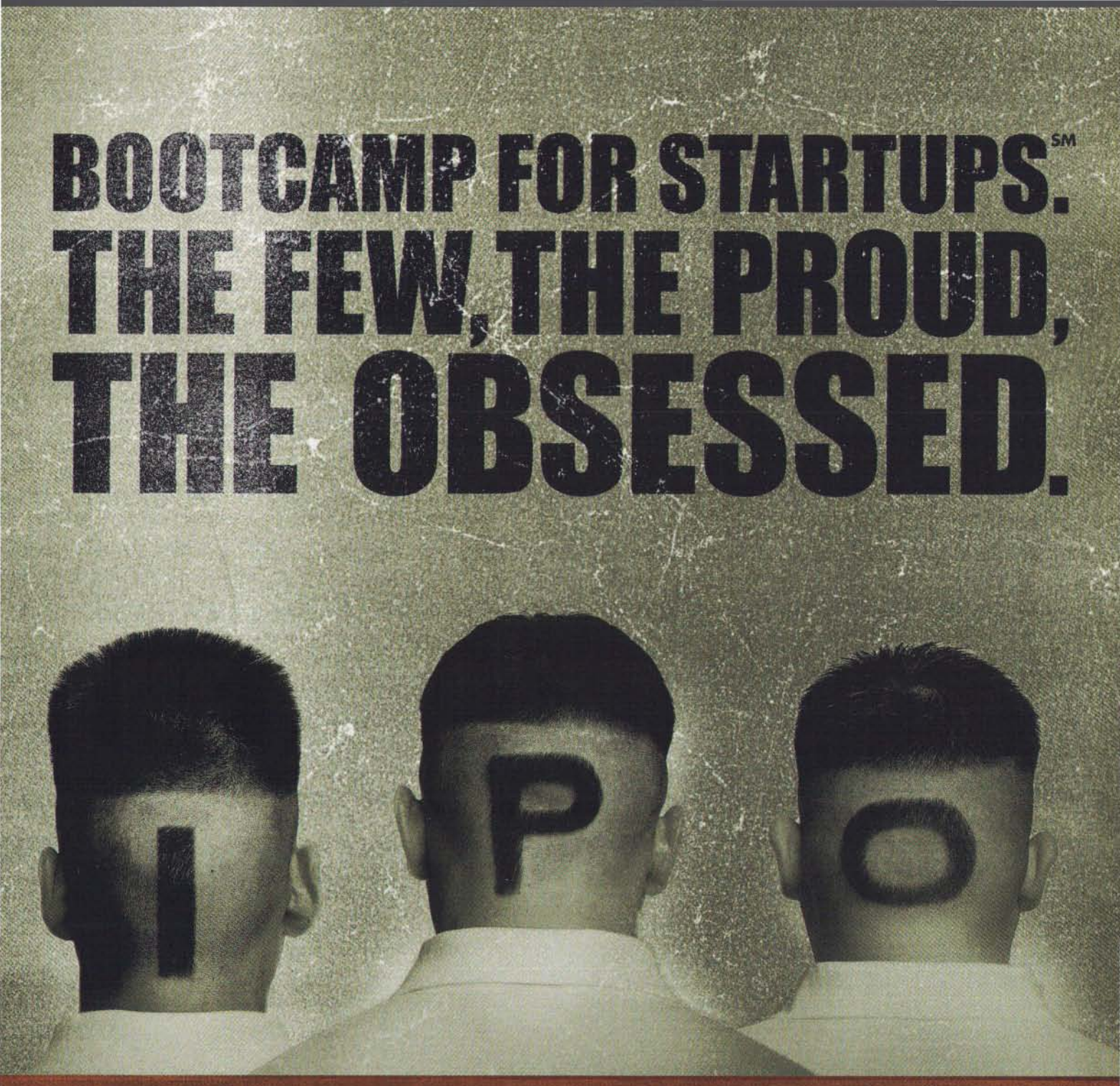
Quesa
<http://quesa.org/>
Quesa - Documentation
<http://quesa.org/reference/docs.html>

Quesa Resources

If you are moving into 3D graphics, there are several places you can start in order to get up to speed. If you are interested in the QD-3D/Quesa approach, you'll get a good overview from a series of articles which originally appeared in Apple's *develop* magazine, and you'll also want to become familiar with Apple's QD-3D documentation, which is available through the QuickTime section of their developer web site. Next, of course, you should take a look at Quesa itself, and download the libraries or source code. There is an active Quesa mailing list, which you'll want to subscribe to, as Quesa is still evolving and there are sure to be discussions of current problems and future directions. You might also want an overview of how QD-3D and OpenGL compare, so that you can make an informed choice about which approach you want to take. You can start by checking out an article "Must-See 3-D Engines" from BYTE Magazine, which compares OpenGL, QD-3D, and Direct3D, and then take a look at the information and resources in a past MacTech Online column from March 1998, written by my predecessor. Both of these are somewhat old and may no longer be accurate in their details, but they'll give you a feel for how the APIs differ in their approaches as well as their consequent strengths and weaknesses. Also, there is an FAQ, as well as several link lists which you can consult to find further information.

QuickDraw 3D: A New Dimension for Macintosh Graphics (June 1995)
<http://www.mactech.com/articles/develop/issue_22/quickdraw.html>
The Basics of QuickDraw 3D Geometries (September 1995)
<http://www.mactech.com/articles/develop/issue_23/thompsonfernicola.html>
New QuickDraw 3D Geometries (December 1996)
<http://www.mactech.com/articles/develop/issue_28/schneider.html>
QuickTime API — QD3D
<http://developer.apple.com/techpubs/quicktime/qtdevdocs/RM/qd3dframe.htm>
Quesa - Mailing List
<http://quesa.org/info/list.html>

BOOTCAMP FOR STARTUPS.℠
THE FEW, THE PROUD, THE OBSESSED.



SILICON VALLEY, October 16–17 **WASHINGTON DC, November 15–16** garage.com™ we start up startups

Attention. Join Garage.com's two-day Bootcamp for Startups. Learn the fundamentals of taking your company from startup to IPO. Hear from the high tech industry's top investors, experts, and entrepreneurs. Gain invaluable information about raising capital, building a buzz, hiring top talent, and launching your product. At ease.

LOG ON TO **WWW.GARAGE.COM/BOOTCAMP** TO LEARN MORE & REGISTER TODAY.

Forbes IBM Wilson Sonsini Goodrich & Rosati Microsoft hp invent

METRIUS ADVANCED TECHNOLOGY plural Tivoli PricewaterhouseCoopers Silicon Valley Bank BOWNE

STARTUPS.COM moto.com ONVIA.com CONXION

Morgan, Lewis & Bockius LLP netpreneur.org San Jose Mercury News The Newspaper of Silicon Valley THESTANDARD SiliconValley.com spacedisk

Must-See 3-D Engines
<http://www.byte.com/art/9606/sec11/art4.htm>
March 1998 MacTech Online
<http://www.mactech.com/articles/mactech/Vol.14/14.03/Mar98MacTechOnline/>
QuickDraw 3D FAQ
<http://www.amplifiedintelligence.com/QD3DFaqContents.html>
Stefan Huber's QD3D Links
<http://www.topoi.ch/3d.html>
Quesa - Links
<http://quesa.org/other/links.html>
QuickDraw 3D Resources
<http://www.designcommunity.com/qd3d.html>

OpenGL Resources

OpenGL itself is an open standard, and the hub of information about this library is at the OpenGL home page. Here you'll find an overview for developers, links to tutorials, and information about the large assortment of books available on OpenGL (including the reference standards *The OpenGL Programming Guide: The Official Guide to Learning OpenGL* and *OpenGL Reference Manual*). As I mentioned above, OpenGL has a large API with a significant learning curve, but it is probably worth the effort if you plan to do hard-core 3D. To make your job a little easier, read up on GLUT, the OpenGL Utility Toolkit. It's a simplified, window-based API which is geared toward those learning OpenGL or using it to writer smaller programs. (GLUT does ship with Apple's OpenGL implementation, by the way.) You can find links to additional resources at the site of the Mesa project, which is an independent library with an OpenGL-compatible API. If you just want to get a flavor for the OpenGL API, try out a recent article on the O'Reilly Network which describes how to use OpenGL to simulate a black hole (really), and also links to additional introductory articles. Finally, keep an eye on Apple's sample code pages for examples of using OpenGL from Cocoa and Carbon, and download the SDK to get you started.

OpenGL Home Page
<http://www.opengl.org/>
Overview of OpenGL
<http://www.opengl.org/developers/about/overview.html>
OpenGL Tutorials and Courses
<http://www.opengl.org/developers/code/tutorials.html>
Books on OpenGL
<http://www.opengl.org/developers/documentation/books.html>
GLUT - OpenGL Utility Toolkit
<http://www.opengl.org/developers/documentation/glut.html>
Mesa - Links
<http://www.mesa3d.org/links.html>
Building a Black Hole With OpenGL
<http://www.oreillynet.com/pub/a/380>
Apple Sample Code - Graphics 3D
<http://developer.apple.com/samplecode/Sample_Code/Graphics_3D.htm>
Apple's OpenGL SDK
<http://developer.apple.com/opengl/index.html>

3D Graphics Foundations

Before you dive into a particular API, you may want to get a feel for the field of 3D graphics in general. If you've never worked with it before, there's quite a bit of conceptual background you'll need, and if you plan to work with it extensively there is also quite a bit of math. The classic text is *Computer Graphics : Principles And Practice* by Foley, Feiner, and Van Dam (ISBN 0201848406), and it is thorough and dense, covering both 2D and 3D graphics. For some lighter reading try the two book's by Jim Blinn, *Jim Blinn's Corner: A Trip Down the Graphics Pipeline* (ISBN: 1558603875) and *Jim Blinn's Corner: Dirty Pixels* (ISBN: 1558604553). They're not a full curriculum but they will give you a feel for some of the interesting and often intricate parts of the field.

3D and Cocoa

Another "interesting" facet of the move to OpenGL for Mac OS X is that we seem to be without a Cocoa-based API for 3D graphics. Certainly, you can use OpenGL from within Cocoa applications—Objective-C was designed to be an extension to ANSI C, so there is no technological barrier to using a C-based API from within a Cocoa application, but it would be more convenient to have a fully object-oriented API to work with, and developers are likely to create their own object-based wrappers for the parts of the API they are using. It would be nice for someone to do this once and for all, and develop a higher-level Cocoa-based library that everyone could use. There is some hope for such an animal as part of the MiscKit project. The MiscKit is a collection of Cocoa-based classes and utilities, assembled under an open-source model before "open source" became a household term. There are all kinds of useful tidbits in the MiscKit, although they are still in the process of being updated for the current Cocoa libraries (as most of the Kit was developed for NEXTStep or OpenStep). Of immediate relevance is something called the 3DKit, which was originally developed by NeXT and later transferred to the MiscKit maintainers (it remains a logically separate project). There appear to be a few licensing issues which may need to be worked out, but with a little luck this could serve as a strong starting point for a high-level 3D framework for Cocoa, or at least as an API model for such a framework.

The MiscKit Frameworks
<http://misckit.org/>

Moving Forward

With OpenGL as a first-class citizen on the Macintosh platform, developers have a real choice of solutions for 3D graphics. It isn't a direct result of the open-sourcing of the core of Mac OS X as Darwin, but open-source projects are becoming more and more relevant (and more an more important) to the platform, and are signaling a cultural shift in the Macintosh community as well as the programming community at large. The Macintosh platform is becoming less and less proprietary, and Macintosh developers are becoming more aware of valuable resources which originated on other platforms, and of their own ability to take part in the process of moving these technologies to the Macintosh. And although it's easy to overlook at first, this is completely in line with the original motivation for the Macintosh, and for Apple as a company: bringing the power of technology to the individual, and letting him make his own choices. Now, more than ever, this means giving this power and choice to the individual developer, as well as to the end user. It's our responsibility to take advantage of this freedom, and participate in driving the platform forward. MT

Compare the best Mac products at the best prices.

MacBuy.com, The Macworld PriceFinder, reviews and compares out-the-door prices of available Macintosh products from a wide array of online vendors. You'll quickly find the best product for your needs, and you'll also find the best price. Shop and compare computers, new and upgrade software, printers and more – it's all there.

The Macworld PriceFinder is your comprehensive resource for choosing the best Macintosh products, getting the lowest prices and buying with confidence from the vendor of your choice.

MACBUY.COM

The Macworld PriceFinder
www.macbuy.com

Continued from page 4

administrator attending his first Summit told me that the presentation on WebSTAR Mail DNS Settings had been worth the price of the Summit in itself. Other presentations in this track were of a more general nature, dealing with XML and Perl and a detailed technical discussion of the Internet's infrastructure. I attended an excellent introduction to XML by developer whose company publishes a server-side XML interpreter. Why wait for browser support? he asked. Why indeed?

4D does the Web, too

Almost a third of the presentations at the Summit dealt with the Web. On the 4D side of the Summit, there were several presentations devoted to the new web features in 4D 6.7 (forthcoming), especially the Web Assistant, the first of the 4D "components." The Web Assistant makes it easier than ever to build Web sites using 4D alone. The keynote showed a demo of another component, a tool for building online stores, code-named "Yapee." (One rumor had it that this is the name of the French developer.) 4D 6.7 supports SSL. A set of extensions for Macromedia Dreamweaver are in development right now, to give 4D developers the ability to use Dreamweaver's outstanding web page-design tools to build pages that will display data dynamically drawn from 4D databases. In our interview, Brendan Coveney told me that 4D, Inc. is deeply committed to the Web's future, which lies with dynamic, database-driven web sites.

Other presentations delved into topics like "4D as a WAP Server" and "e-Commerce with 4D." The latter was presented by the maker of Web Server 4D (WS4D), a remarkable off-the-shelf web serving and e-commerce application which proves almost better than anything else how powerful and flexible 4D's programming language is: WS4D—in many ways a competitor of 4D now—is itself programmed entirely in 4D! In the "beginner's track," I presented a well-attended session on using 4D as a backend for Lasso-driven sites. *Not* in the beginner's track, 4D maestro David Adams gave an advanced full-day seminar in 4D Web techniques after the main part of the Summit.

4D 6.7 and OS X

In the keynote, Brendan Coveney played a snazzy little game called Time Matrix, written in 4D by the folks at DataCraft. (DataCraft is the publisher of Foundation, a brilliantly designed shell widely used by 4D developers.) Time Matrix was run first under OS 9, then it was run again, under OS X beta. In view of the obvious complexity of the underlying code, the remarkable thing was not that the second demo sported the Aqua look, but that not a single line of code had to be rewritten.

The folks at 4D, Inc. are committed to (if not downright obsessed with) making sure that old systems don't break when new ones are released. I was assured by several different developers on different occasions that it is possible to open a 1987-vintage version 1.0 database in the year 2000 under 4D version 6.5 and that it will in all likelihood run fine with few or no changes. After demonstrating that the core 4D application itself will make the transition to OS X without a hitch, Coveney went on to tell developers that 4D, Inc. is working very hard to make sure that it is as easy as possible for plug-in developers to port their products to OS X.

AreaList Pro is dead. Long live PowerView!

I am not an old-enough hand with 4D to know the story first-hand, but I have heard it many times from experienced developers. They sit down and get a far-away look when they start to tell you about it, the way veterans do before talking about The War. It goes something like this: There used to be a third-party plug-in for 4D called AreaList Pro, which was relied upon by every serious 4D developer. AreaList provided a set of life-saving functions not built into 4D, all of them derived from its central trick of displaying arrays on screen. AreaList was a VBD (very big deal).

Then one dark day, the company that had been publishing AreaList and several other crucial 4D tools decided that its own business plan no longer included 4D and that it would stop developing and supporting these tools. To hear the old-timers tell it, it was like waking up tomorrow to discover that you could not buy gasoline for your car—anywhere.

At last year's Summit in Chicago, 4D, Inc. promised developers that it would solve the problem caused by AreaList's death, and this year, they delivered on the promise by announcing PowerView, a new tool in 6.7 which combines the features of ALP and 4D Chart (4D's spreadsheet plug-in). The demo of PowerView showed it to be fast and flexible. One part of the demo consisted of a ballet of formatted table cells that Brendan Coveney had to assure the audience had *not* been done in Flash! Several developers I talked to thought that, while the preview of 4D running under OS X was good news, the announcement of a replacement for AreaList Pro was the news that mattered most to them.

To me as a rookie 4D user still spending most of my time on the bench, it was interesting to discover that there are still companies brave enough to *make* promises and at least equally interesting to see a company actually *keep* them.

Everything Else

The rest of the program was nicely diversified. One presenter in the WebSTAR track warned his audience that his talk was going to get a bit geeky. He needn't have bothered. The entire conference was unashamedly geeky. The level of discourse among the attendees was consistently high. Even the beginners track included presentations likely to make expert FileMaker users like me sweat a little, such as "Parameter Passing and Generic Code," "Accelerated Text Parsing with BLOBs," "Pointers on Pointers," and "Multi-Process Programming." Other presentations on 4D were similarly diversified, dealing with memory management, interprocess data transfer, and localization of applications. I was not able to attend the latter, but its presence on the program reminded me of 4D's international character. 4D, Inc. in the U.S. is a wholly-owned subsidiary whose parent country is in France. 4D has long provided extraordinary support for international users, including full-support for double-byte languages like Chinese and Japanese.

A few of the sessions were bleeding edge. I attended a session on using speech-recognition and synthesis as a replacement for the conventional UI. The session was fascinating, especially when the presenter's demo behaved as expected. I left feeling that I might personally wait a year before worrying about this subject again.

Because you have complete control over the UI (including menus) and because you can compile your code into double-clickable programs, 4D is a great tool for developing vertical market applications. A couple panel sessions provided detailed advice for commercial developers from those who have already been there and done that. I personally learned that we at Polytrope have been doing almost everything wrong.

Windows?

The casual observer could easily have gotten the impression that this was a convention attended exclusively by Macintosh users. C.K. Hahn, Senior Director, Developer Technical Services, Apple Computer, Inc., spoke briefly during the keynote to give Apple's blessing on 4D, Inc.'s commitment to OS X. Apple sponsored the wonderful Internet café for Summit attendees, complete with an Airport base station. Many of the sessions dealt specifically with the Mac OS. My completely unscientific guess is that a good eighty percent or more of the attendees would consider themselves primarily Mac OS users. And yet 4D is a cross-platform product. More than that: Brendan Coveney told me that roughly 75% of their sales are for the Windows platform (mainly 4D Server for NT boxes)! This paradox leads me to two observations. First, it appears that developing 4D databases on the Mac and deploying them under Windows is easy and reliable. If there were a lot of problems in this arrangement, I would have expected to see at least a couple presentations like "Pot-holes to avoid with the Windows compiler" or "Memory Management under Windows NT." Second, the market — not just 4D, Inc.'s market, but *my* market, the developers' market — is Windows.

The Big News

On the last day, I asked everybody I talked to what the big news of the Summit was. Some said it was the acquisition of WebSTAR by 4D, Inc. Some mentioned PowerView. Some pointed to the web features of the forthcoming 4D 6.7. Many talked with excitement about seeing 4D running under OS X. But when I asked Brendan Coveney what the big news was, he did not hesitate to give me the low-tech answer that I think is the best of all: "The big news of the Summit is that the 4D community is alive and growing and the atmosphere is tremendously positive."

Credits and More Information

Many thanks to those credited above and to many others not credited for speaking to me. Special thanks to John Steele of Elucidata in Fort Worth for his clarifications with regard to AreaList Pro. The official 4D Summit web site is at <http://www.4DSummit.com>, but you need a password to get into the really useful pages. No password is required to get into developer and Summit presenter Bryan Green's unofficial celebration of the Summit: <http://www.4DSummitnotes.com>. MT

By Bruce Lawton

Talking to Your Home

Introduction

The futuristic world of talking to your home to operate everything in it is here now! Well, how many times have we heard claims like that? The truth is: for some it really is here now; for others, it is not quite ready yet. Speech recognition technology is delivering some amazing results already and it is just getting better as the computers get faster and the recognition software continues to improve.

To make it happen, we'll be talking about voice commands, microphone(s), a Power Macintosh, software, X-10 home automation hardware and your existing lamps, appliances and other gadgets. This discussion assumes you're already familiar with general X-10 home automation. If not, check out the June 2000 issue of MacTech.

Why You Might Want to Use Voice Commands

The most useful application for speech recognition is for persons wanting to overcome a physical handicap. With limited mobility or limited dexterity, speech recognition enables someone to operate lamps, appliances and home entertainment units that would otherwise be difficult or impossible to operate. In this way, the quality of life can be greatly improved, giving the person a bit more independence.

For those who are not physically limited, operating the home by voice can be a matter of convenience or just plain fun. How many times have you settled into bed and then realized (perhaps with a little helpful reminder from your dear significant other) that some lights were left on? The easiest thing to do is to say "Turn off all lights."

It is also impressive to show your PC friends what your Macintosh will do in response to your voice commands. There's nothing quite like seeing everyday hardware operated by voice.

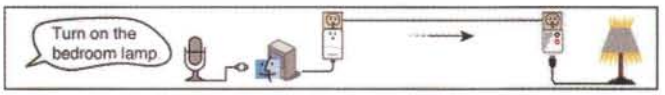


Figure 1. *From voice to lamp.*

Setting up the Microphone(s)

To get started, you must first plan how you're going to get your voice to your Macintosh. There are at least three ways to do it. The most inexpensive microphone configuration is the standard PlainTalk, corded microphone connected directly to your Mac. The obvious drawback is that the cord restricts the distance you can go from your Mac while issuing voice commands. In some situations, this may be adequate.

The most cost-effective approach giving you more range is to use a wireless microphone that transmits your voice as your own personal FM radio station. Then you have a small FM radio connected to your Mac to feed the sound in. The range on the microphone should enable operation from anywhere in a normal-sized home. My

Bruce Lawton is still an enthusiastic Macintosh developer who first got hooked over 15 years ago. He can be reached at bruce@alwaysthinking.com.

own experience is that the Pro.2 microphone from Lotus Productions (www.lotusproductions.com, (800) 211-3778) works well and costs around $45. A small $25 radio from Radio Shack will receive the FM signal and should be connected to either your A/V Mac's RCA input jacks or directly to the PlainTalk mic jack using a standard 3.5 mm stereo plug as shown below. The PlainTalk jack is really a stereo line-level input. The extra length of the plug allows the Mac to supply power to the tip which is then used by circuitry in the PlainTalk mic. The normal 3.5 mm stereo plug doesn't use that available power. The headphone jack of the radio provides an audio signal compatible with the PlainTalk mic input. Make sure you adjust the radio headset volume to get the clearest sound with no clipping.

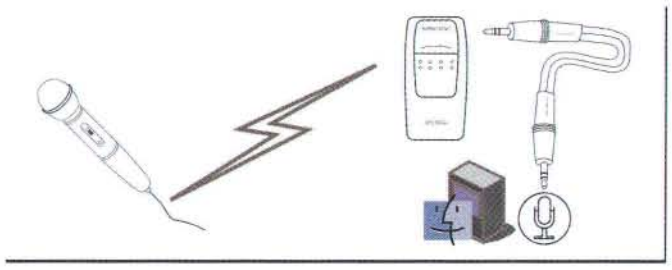


Figure 2. *FM microphone and radio.*

The deluxe approach is to use in-wall microphones and a mixer. This costs the most, but can give you completely hands-free voice control from any room. There is a fair amount of experience in the comp.home.automation newsgroup from those who have set this up on Windows. It can work well, though I must admit I haven't done this myself — my own home has far too much "noise". Wired microphones can be great for a home with one or maybe two people living there, but may not work well with several people. The Mac's PlainTalk can get confused if it is listening to many voices at once. When using in-wall microphones, each mic has a cable that is run back to the mixer, which is presumably near the Mac. The audio output from the mixer is then connected to the Mac's sound input. Similar to the FM radio configuration, you can feed the line-level signal either into your A/V Mac's RCA input jacks or directly into the PlainTalk mic jack using a standard 3.5 mm stereo plug.

Shure (www.shure.com, (847) 866-2200) makes the SCM-410 and SCM-810 mixers with 4 and 8 inputs respectively. They dynamically cut out the microphones that are not being used, improving the audio quality for the microphone you're speaking to. They cost $800 and $1,200 at iAutomate (www.iAutomate.com, (800) 741-6790). HomeVoice also makes a mixer, sold as part of their "Multiroom Processor Kit" at SmartHome (www.smarthome.com, (800) 762-7846) for $690. Good, compatible, in-wall microphones vary greatly in price, from $70 to $250 or more. One of the most recommended models is the Crown PZM-11, sold at iAutomate and SmartHome.

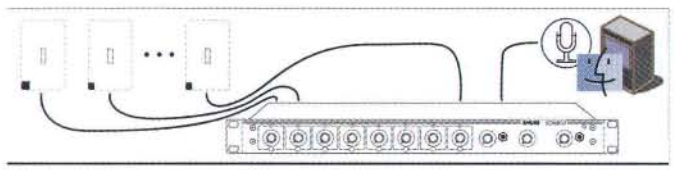


Figure 3. *In-wall microphones with a mixer.*

The Software

For our purposes, there are two categories of speech recognition software: dictation and discrete commands. IBM's ViaVoice is an excellent way to get your speech dictation into the Macintosh, but the resulting stream of words is not "understood" by the ViaVoice software. That is, it doesn't know what action you want taken in response to the words. Dictation software is intended as an alternate method of text input — it is not going to help you operate your X-10 system. This is where you want software that

understands discrete commands. PlainTalk is at the heart of this type of speech recognition. As it has steadily improved over the years and CPU's have sped up, the reliability has become quite good. The Thinking Home application uses PlainTalk to make your X-10 devices available. First, install PlainTalk. This is done via a "Custom" installation from your MacOS CD or it can be downloaded from Apple's web site at <http://www.apple.com/macos/speech/>. Next, ensure you have checked the "Respond to Voice Commands" checkbox in the Preferences as seen in **Figure 4**.

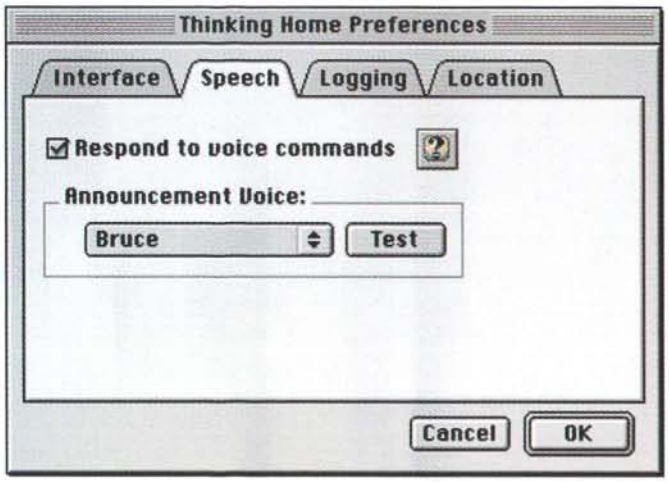


Figure 4. *Enabling speech commands.*

You need to have a device named "Bedroom Lamp" like the example of **Figure 5**. Now you can say "Turn off the bedroom lamp." and Thinking Home will do the right thing.

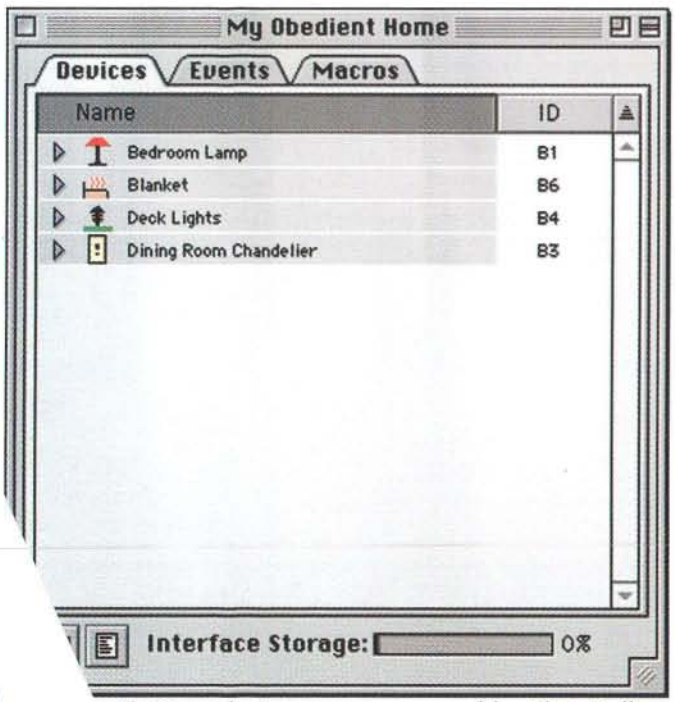


Figure 5. *Your device names are used by PlainTalk.*

Syntax of What You Say to Make it Work

In order to operate a device using Thinking Home, you speak a normal sentence, with the action followed by the device name. For example, "Turn off the bedroom lamp." or "Set the thermostat to heat." The precise syntax is shown here in Backus-Naur Form notation:

```
<voice-command> ::= <action> <device-name> |
      "What is the status of the> <device-name> |
    "Set the" <thermostat-name> "to" <mode>
<action>  ::= "Turn on" | "Turn off" | "Dim" |
"Brighten"
<device-name> ::= the device name used in the
Thinking Home document
<thermostat-name>::= <device-name> | "thermostat"
<mode>  ::= "Heat" | "Cool" | "Off"
```

More examples:

Turn on the bedroom lamp.
Turn off the kitchen light.
Dim the theater lights. — will be dimmed by 35%
Brighten the bedroom lamp. — for a gentle start in the morning.
Set the thermostat to cool.
What is the status of the sidewalk lights?

Note that the thermostat can be referred to by the name you gave it, like any other device. Or you can refer to it simply as "thermostat" and Thinking Home will use the first thermostat it finds in any open document.

There are some caveats to asking for the status of a device. First of all, the Mac will speak the reply and you may not be at your Macintosh if you are using the wireless mic or mics installed in each room. The other issue is the accuracy of the status. This will depend on whether the X-10 module is able to report its status. Two-way device modules ($25+) will report their status, but the most common modules that sell for $4 to $12 do not. If you're using an ActiveHome CM-11A computer interface, it will keep track of X-10 commands observed on your home wiring and will report the status of the lamp. This assumes you have not operated it manually. The report from a two-way lamp module would reflect its actual state, including manual operation.

You can also control your X-10 thermostat by saying "Set the thermostat to cool." The choices for thermostat modes are cool, heat and off. Unfortunately, you cannot say "Set the thermostat to 72 degrees." This is one place where PlainTalk is not quite ready. Recognizing spoken numbers is much more difficult than most other speech.

Speakable Items (installed with PlainTalk) is another way to access some of Thinking Home's functionality via speech. For example, you may want to know the time of sunset on a regular basis. By making a Speakable Item like the example below, you can say to your Macintosh "When is sunset?" and the Speakable Items application

will run the the AppleScript by that name. The AppleScript can, in turn, tell Thinking Home to speak the time of today's sunset as seen here in **Figure 6**.

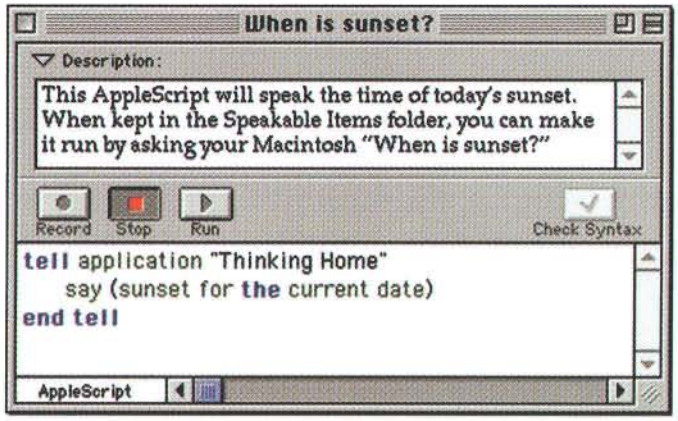


Figure 6. *Using Speakable Items to get the time of sunset.*

This is also how you can get XTension v3.0 (from Sand Hill Engineering) to respond to your voice. Although XTension does not have built-in support for PlainTalk, it does have excellent support for AppleScript. You can use Speakable Items for speech recognition and make an AppleScript for each command you want recognized. Your scripts in the Speakable Items folder will tell XTension which X-10 device to operate. For example, to get the time of sunset, you'd write a script almost exactly as that for Thinking Home.

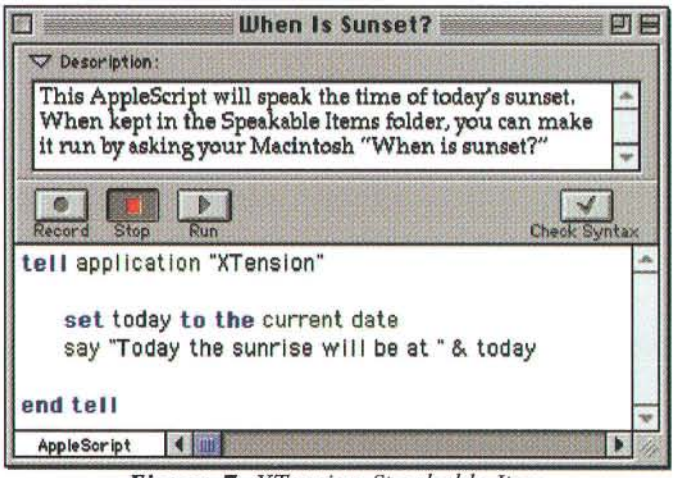


Figure 7. *XTension Speakable Item for getting the time of sunset.*

To use XTension to operate devices via speech, you create an AppleScript like that in **Figure 8** and name it as you intend to speak it. Like before, you name this script "Turn off the bedroom lamp". You'll also need to make a similar one for "Turn on the bedroom lamp."

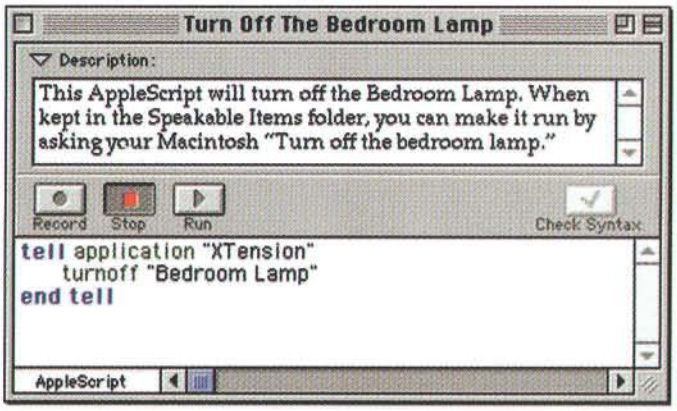


Figure 8. *XTension Speakable Item for operation of a device.*

To get device status using XTension, you write an AppleScript similar to that shown in **Figure 9**.

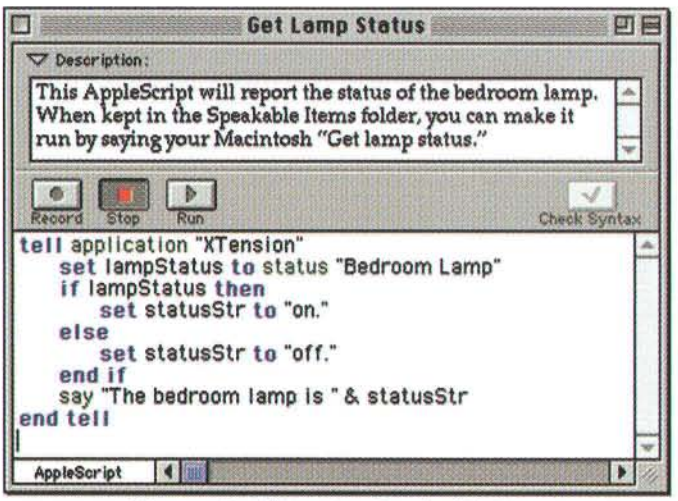


Figure 9. *XTension Speakable Item for getting the status of a device.*

Tips

Since faster CPU's seem to perform better than slower ones, you will probably want to avoid using a Power Macintosh running less than 100 MHz.

The Shure web site has a wealth of useful information and general advice regarding the various options, such as ceiling - vs. wall-mounted microphones. <http://www.shure.com/support/technotes/app-soundcard.html#Macintosh>

Before you demo your cool home to others, test it out in advance. In particular, check that it works reliably from all locations you plan to be when showing it. When using a wireless microphone, ensure the range is sufficient to operate consistently from all locations where you will be using it. Also, make sure the ambient sounds are the same. Putting on background music just as your guests arrive will change the "noise" level and your speech commands may very well get lost. Once you have it set up nicely, your friends will be amazed.

Conclusion

Integrating speech recognition into your Macintosh X-10 home automation can be useful, convenient and fun. This futuristic technology is ready and affordable today and still improving. Select an arrangement that fits your lifestyle and budget and try it out.

Links

- http://www.apple.com/macos/speech/
  Apple's Plaintalk download page.
- http://www.shed.com/
  Sand Hill Engineering home page (for XTension)
- http://www.alwaysthinking.com/products/products.html
  Thinking Home product page.
- http://www.lotusproductions.com
  Lotus Productions (wireless microphones)
- http://www.shure.com/support/technotes/app-soundcard.html#Macintosh
  Shure home page (audio mixers)
- news:comp.home.automation
  Home automation newsgroup with emphasis on X-10.

MT

United States Postal Service

Statement of Ownership, Management, and Circulation

1. Publication Title	2. Publication Number	3. Filing Date
MacTech Magazine	1067-8360	10/1/00
4. Issue Frequency	5. Number of Issues Published Annually	6. Annual Subscription Price
Monthly	12	$47.00

7. Complete Mailing Address of Known Office of Publication (Not printer) (Street, city, county, state, and ZIP+4): P.O. Box 5200, Westlake Village CA 91359 — Contact Person — Telephone (805) 494-9797

8. Complete Mailing Address of Headquarters or General Business Office of Publisher (Not printer): Same as above

9. Full Names and Complete Mailing Addresses of Publisher, Editor, and Managing Editor (Do not leave blank)

Publisher (Name and complete mailing address): Neil Ticktin - address above

Editor (Name and complete mailing address): n/a

Managing Editor (Name and complete mailing address): Jessica Stubblefield - address above

10. Owner (Do not leave blank. If the publication is owned by a corporation, give the name and address of the corporation immediately followed by the names and addresses of all stockholders owning or holding 1 percent or more of the total amount of stock. If not owned by a corporation, give the names and addresses of the individual owners. If owned by a partnership or other unincorporated firm, give its name and address as well as those of each individual owner. If the publication is published by a nonprofit organization, give its name and address.)

Full Name	Complete Mailing Address
Xplain Corporation	PO Box 5200 Westlake Village CA 913
Neil Ticktin	same as above
Andrea Sniderman	same as above

11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages, or Other Securities. If none, check box → X None

Full Name	Complete Mailing Address

12. Tax Status (For completion by nonprofit organizations authorized to mail at nonprofit rates) (Check one)
The purpose, function, and nonprofit status of this organization and the exempt status for federal income tax purposes:
X Has Not Changed During Preceding 12 Months
☐ Has Changed During Preceding 12 Months (Publisher must submit explanation of change with this statement)

PS Form 3526, October 1999 (See instructions on Reverse)

13. Publication Title: MacTech Magazine	14. Issue Date for Circulation Data Below: August 2000	
15. Extent and Nature of Circulation	Average No. Copies Each Issue During Preceding 12 Months	No. Copies of Single Issue Published Nearest to Filing Date
a. Total Number of Copies (Net press run)	8955	9249
b. Paid and/or Requested Circulation (1) Paid/Requested Outside-County Mail Subscriptions Stated on Form 3541. (Include advertiser's proof and exchange copies)	5032	5145
(2) Paid In-County Subscriptions Stated on Form 3541 (Include advertiser's proof and exchange copies)	0	0
(3) Sales Through Dealers and Carriers, Street Vendors, Counter Sales, and Other Non-USPS Paid Distribution	1909	2764
(4) Other Classes Mailed Through the USPS	24	6
c. Total Paid and/or Requested Circulation (Sum of 15b. (1), (2),(3),and (4))	6965	7915
d. Free Distribution by Mail (Samples, complimentary, and other free) (1) Outside-County as Stated on Form 3541	0	0
(2) In-County as Stated on Form 3541	0	0
(3) Other Classes Mailed Through the USPS	44	35
e. Free Distribution Outside the Mail (Carriers or other means)	547	510
f. Total Free Distribution (Sum of 15d. and 15e.)	591	545
g. Total Distribution (Sum of 15c. and 15f)	7556	8460
h. Copies not Distributed	1399	789
i. Total (Sum of 15g. and h.)	8955	9249
j. Percent Paid and/or Requested Circulation (15c. divided by 15g. times 100)	92	94

16. Publication of Statement of Ownership
☐ Publication required. Will be printed in the November 2000 issue of this publication. ☐ Publication not required.

17. Signature and Title of Editor, Publisher, Business Manager, or Owner: X [signature] Andrea Sniderman, President — Date: 10/2/00

I certify that all information furnished on this form is true and complete. I understand that anyone who furnishes false or misleading information on this form or who omits material or information requested on the form may be subject to criminal sanctions (including fines and imprisonment) and/or civil sanctions (including civil penalties).

Instructions to Publishers

1. Complete and file one copy of this form with your postmaster annually on or before October 1. Keep a copy of the completed form for your records.
2. In cases where the stockholder or security holder is a trustee, include in items 10 and 11 the name of the person or corporation for whom the trustee is acting. Also include the names and addresses of individuals who are stockholders who own or hold 1 percent or more of the total amount of bonds, mortgages, or other securities of the publishing corporation. In item 11, if none, check the box. Use blank sheets if more space is required.
3. Be sure to furnish all circulation information called for in item 15. Free circulation must be shown in items 15d, e, and f.
4. Item 15h., Copies not Distributed, must include (1) newsstand copies originally stated on Form 3541, and returned to the publisher, (2) estimated returns from news agents, and (3), copies for office use, leftovers, spoiled, and all other copies not distributed.
5. If the publication had Periodicals authorization as a general or requester publication, this Statement of Ownership, Management, and Circulation must be published; it must be printed in any issue in October or, if the publication is not published during October, the first issue printed after October.
6. In item 16, indicate the date of the issue in which this Statement of Ownership will be published.
7. Item 17 must be signed.

Failure to file or publish a statement of ownership may lead to suspension of Periodicals authorization.

PS Form 3526, October 1999

ADVERTISER/ PRODUCT LIST

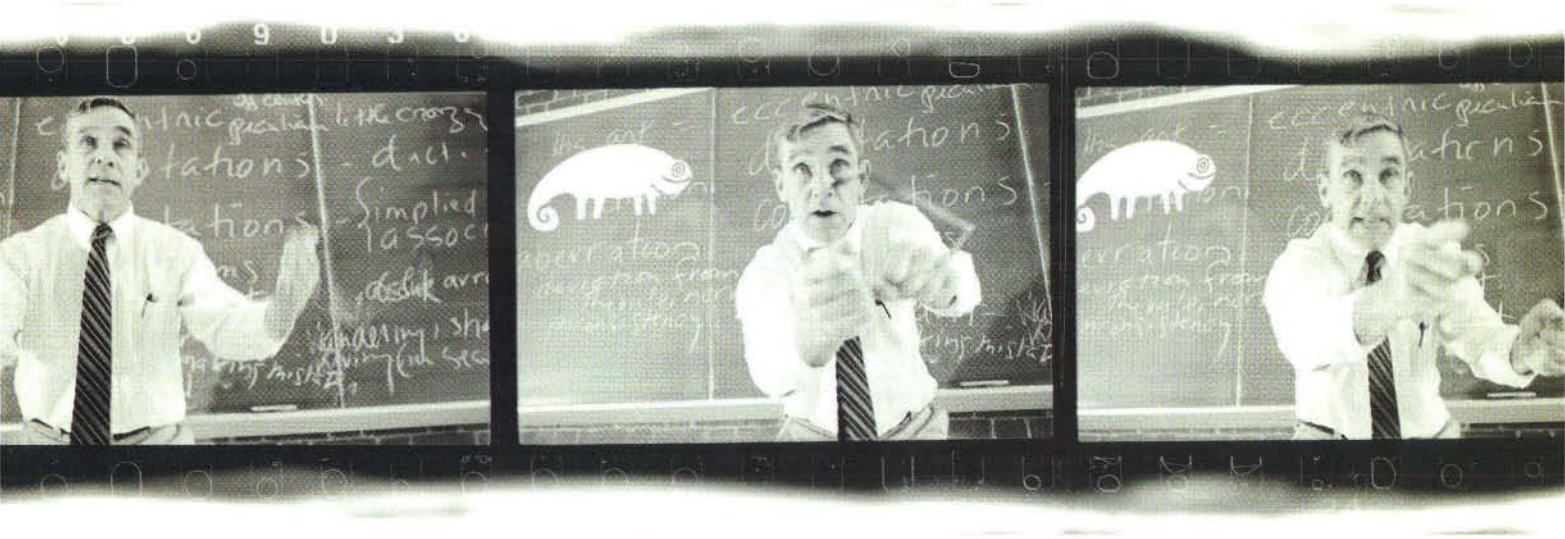
List of Advertisers

Mac OS X Porting and Development Showcase

List of Products

Mac OS X Porting and Development Showcase

HOW DO YOu SAY SuPERB IN LINuX?



SuSE.

{soo´ sah} is {soo´ pûrb}

When you think of Superb, you think of unusually high quality. Like a superb wine, for example. Majestic. Rich. Luxurious. Superior. ■ SuSE Linux is all that. And more. More experience. More adaptability. More applications—over 1500. ■ More power to you. And more freedom, too. ■ No wonder more than 50,000 enterprises worldwide bank on its superb open source solutions. Making SuSE the international Linux leader—setting a higher standard for excellence, simplicity and support. ■ Even the price is superb. ■ So, how do you say superb in Linux? SuSE. It's a lesson well learned.

www.SuSE.com

The freedom to change.
The power to change the Linux world.

SuSE

Versions for Intel, Alpha, and PowerPC

Now Shipping! CodeWarrior for Mac OS & Mac OSx

Keeping Mac dreams alive since 1993.

Two great operating systems, one CodeWarrior.

Mac developers depended on CodeWarrior to make the platform architecture shift from 68K to PowerPC processors. Now CodeWarrior does it again, speeding your transition to the next new operating system. CodeWarrior for Mac OS, Version 6.0 supports development for both OS X and Classic Mac operating systems from a single, powerful, award-winning Integrated Development Environment. Discover how CodeWarrior for Mac OS, Version 6.0 can help you realize your Mac development dreams.

Visit **www.metrowerks.com/go/mac**.

CodeWarrior®

metrowerks®
Software Starts Here